الجمهـــوريـــــة الجزائـــريـــــة الديمقـــراطيـــة الشـعبيـــة

République Algérienne Démocratique et Populaire

وزارة التـــعلـــيـــــم العـــــالـــــي والبـــحـــــث العـلمـــي

Ministère de l'Enseignement Supérieure et de la Recherche Scientifique

جامعة وهران 2 محمد بن أ حمد

Université Ahmed Ben Mohamed 2 d'Oran

معهد الصيانة و الأمن الصناعي

Institut de Maintenance et de Sécurité Industrielle

Département de Sécurité Industrielle et Environnement

MÉMOIRE

Pour l'obtention du diplôme de Master

Filière : génie Industrielle

Spécialité : instrumentation dans maintenance industriel

Thème

Conception d'un simulateur ladder Sous environnement delphi

Présenté et soutenu publiquement par :

Noms :**Mehali** Prénoms : **Farid**

Haloui **Youcef Ouail**

Devant le jury composé de :

Encadreur : **dr adda neggaz samir**
President : **dr mekki Ibrahim al khalil**
Examinateur : **dr reguige**

2022\2023

Remerciements

À ma famille, mes amis et mes professeurs,

Je dédie ce projet de fin d'études à tous ceux qui m'ont soutenu, encouragé et accompagné tout au long de mon parcours universitaire. Je remercie du fond du cœur ma famille, qui a toujours cru en moi et qui m'a offert les meilleures conditions pour réussir mes études. Je remercie également mes amis, qui ont partagé avec moi des moments inoubliables et qui ont été présents dans les moments difficiles. Je remercie enfin mes professeurs, qui m'ont transmis leurs connaissances, leurs conseils et leur passion pour leur domaine. Grâce à eux, j'ai pu réaliser ce projet qui représente l'aboutissement de mes études et le début d'une nouvelle étape dans ma vie professionnelle.

Ce projet n'aurait pas été possible sans l'aide et la collaboration de plusieurs personnes que je tiens à remercier ici. Je remercie tout d'abord mon tuteur de projet, MR adda neggaz samir , qui m'a guidé, orienté et conseillé tout au long de la réalisation de ce travail., Je remercie enfin tous les participants à mon étude, qui ont accepté de donner de leur temps et de leur attention pour répondre à mes questions.

Je suis fier du travail accompli et j'espère qu'il apportera une contribution utile au domaine de la recherche sur le sujet conception du sumulateur a l'nvirenment delphi. Je souhaite également exprimer ma gratitude à l'université institut de maintenance industriel, qui m'a offert la possibilité de suivre une formation de qualité et de développer mes compétences et mes connaissances.

Merci à tous !

Haloui youcef ouail

Remerciements

À mon projet de fin d'études, symbole de détermination et de réussite, je dédie ce moment avec reconnaissance.

À mes parents, source inépuisable d'amour et de soutien, je vous offre ce succès avec tout mon cœur.
À ma femme , complice de cette aventure, je partage ce triomphe avec tendresse.
À mes chers frères, et ma chère sœur.
mes amis, ma famille, qui ont illuminé ce parcours, je vous remercie pour votre présence constante.
À mon binôme, partenaire de défis et de succès, ce projet est le fruit de notre collaboration exceptionnelle.
À mes enseignants, guides précieux de mon apprentissage, je vous offre ce projet en signe de gratitude pour votre enseignement inspirant.

Mr. MEHALI FARID

Liste des tableux et figure :

Introduction générale

Cette introduction décrit l'évolution des systèmes automatisés industriels en trois phases, passant de la vision imitative des automates à une logique mécaniste, puis à une approche systématique basée sur la cybernétique et la théorie des systèmes. Il mentionne les automates programmables industriels (API) modernes, qui remplacent les systèmes plus anciens grâce à des micro-contrôleurs interconnectés. L'utilisation de langages de programmation comme le langage à relais LADDER est évoquée, ainsi que l'importance des interfaces graphiques et des logiciels de supervision dans la programmation des automates. Finalement, il souligne la tendance actuelle des concepteurs à se concentrer sur le développement de logiciels pour des applications automatisées plus efficaces, avec une augmentation des coûts associés à mesure que ces logiciels deviennent plus complexes et sophistiqués.

Dans le premier chapitre de notre étude, nous avons exploré en profondeur l'univers des systèmes automatisés, ces innovations qui ont autrefois captivé notre imagination et suscité l'admiration des passionnés, mais qui sont aujourd'hui devenus des éléments presque invisibles dans notre environnement industriel quotidien.

Le deuxième chapitre nous a transportés dans le monde passionnant des automates programmables industriels (API), également connus sous le nom de Programmable Logic Controllers (PLC). Nous avons examiné leur rôle essentiel dans l'automatisation de divers processus industriels, remplaçant avantageusement les systèmes automatisés plus anciens et encombrants.

Au troisième chapitre, nous nous sommes plongés dans l'univers du langage Ladder, un élément fondamental de la programmation des automates. Nous avons exploré son origine qui remonte à la norme de 1969 et comment il facilite la transition de la logique câblée à la logique programmée.

Le quatrième chapitre nous a emmenés dans le domaine du langage Pascal et de l'environnement de développement Delphi 7. Nous avons découvert comment ces outils puissants sont utilisés pour concevoir des programmes, notamment notre propre projet de programmation orientée objet pour un programme lader.

Ainsi, à travers ces différents chapitres, notre étude a tracé le parcours passionnant de l'évolution des systèmes automatisés jusqu'à notre plongée dans le monde complexe des automates programmables industriels, des langages Ladder, et enfin, des langages Pascal et Delphi.

Chapitre I

Les systèmes automatisés

I. Systèmes automatisés :

I.1 - Fonction Globale d'un Système :

L a fonction globale de tout système (figure I.1) est de conférer une valeur ajoutée, à un ensemble de matières d'œuvre dans un ou un contexte donné. De plus, un système de production est dit « industriel » si l'obtention de cette valeur ajoutée, pour un ensemble de matières d'œuvre donné, a un caractère reproductible et peut être exprimée et quantifiée en termes économiques.

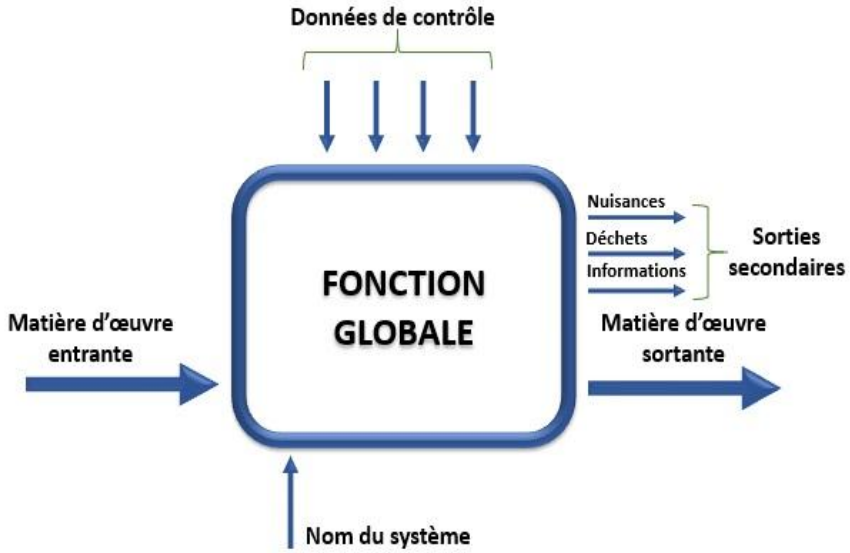


Figure I.1 : fonction globale d'un système

I.1 Définitions de l'automatisme :

Un **automatisme** est un sous-ensemble d'une machine, destinée à remplacer l'action de l'être humain dans des tâches en générales simples et répétitives, réclamant précision et rigueur. On est passé d'un système dit manuel, à un système mécanisé, puis au système automatisé.

(Voire des exemples de systèmes automatisés dans l'annexe)

Dans l'industrie, les automatismes sont devenus indispensables : ils permettent d'effectuer quotidiennement les tâches les plus ingrates, répétitives et, dangereuses. Parfois, ces automatismes sont d'une telle rapidité et d'une telle précision, qu'ils réalisent des actions impossibles pour un être humain. L'automatisme est donc synonyme de productivité et de sécurité.

Le savoir-faire de l'opérateur est transposé dans le système automatisé, il devient le PROCESSUS. (Voire annexe)

I.2 Introduction à l'automatisation :

L'automatisation consiste à < rendre automatique > les opérations qui exigeaient auparavant l'intervention humaine.

Une autre définition :

L'automatisation de la production consiste a transférer tout, ou partie des taches de coordination, auparavant exécuter par des opérateurs humains, dans un ensemble d'objets techniques appelé **PARTIE COMMANDE.**

La partie commande mémorise **le SAVOIRE FAIRE** des opérateurs pour obtenir la suite des actions a effectuer sur les **matières d'œuvre** (voire annexe) afin d'élaborer **la valeur ajouter** (voire annexe) .

Elle exploite un ensemble d'informations prélevées sur la partie Opérative pour élaborer la succession des ordres nécessaires pour obtenir les actions souhaitées.

I.3 Buts (ou objectifs) de L'automatisation :

Objectifs : La compétitivité de l entreprise et des produits.

Cette compétitivité passe par la qualité, la maîtrise des coûts et L'innovation. Cela induit une disponibilité à tous les niveaux. On cherche donc à améliorer la productivité. L'amélioration des conditions de travail, et surtout la sécurité, fait partie des objectifs de l'automatisation

Les buts (ou objectifs) de l'automatisation sont donc :

Éliminer les tâches répétitives,

- Simplifier le travail de l'humain
- Améliorer la flexibilité de production
- Améliorer la qualité du produit grâce a une meilleure respectabilité de valeur ajouté
- S'adapter a des contextes particuliers :

  -adaptation a des environnements hostiles pour l'homme (milieu salin, spatial nucléaire…)

  -adaptation a des taches physiques ou intellectuelles pénibles pour l'homme (manipulation de lourdes charges, tâches répétitives parallélisées …)
- augmenter la sécurité, etc.…

D'autres objectifs, a caractères sociaux, financiers… peuvent s'ajouter a ceux-ci .

I.4 Conduite et surveillance d'un système automatisé

Il s'avère très difficile en pratique d'intégrer dans une partie commande la totalité des savoir-faire humains de sorte que l'automatisation reste souvent partielle 0: certaines taches restent à des humaines.

A ces causes techniques viennent s'ajouter des considérations économiques de compétitivité, des considérations financières important un fractionnement des investissements des considérations sociales d'automatisation douce.

Certaines taches restent donc manuelle et l'automatisation devra donc prendre en compte la spécialité du travail humain, c'est-à-dire RETUNES par le concepteur parmi en ensemble de situations possibles, Or il est impératif de pouvoir faire face à des situations NON PREVUES (donc non retenues en générale pour des raisons économiques compte tenu de leur faible probabilité)

Seul un opérateur peut alors intervenir et prendre les décisions requises par cette situation : il assure une fonction de CONDUITE ct de SURVEILLANCE du système automatisé. Cette fonction peut être plus ou moins assisté par un ensemble de moyens (pupitres, informatique…)

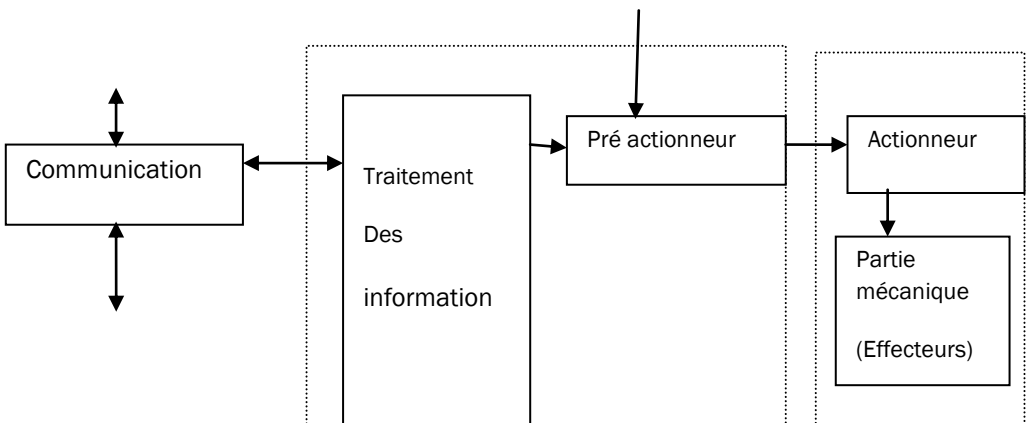
Le concepteur devra alors :

-fournir à l'intervenant (ou lui permettre de prélever) toutes les informations SIGNIFICATIVES (ou INDICES) nécessaires à l'analyse de la situation,

-lui permettre d'agir sur le système, soit directement (dépannage…), soit indirectement (consignes de sécurité, de marches et d'arrêts …)

I.5 Structure de système automatisé :

La structure générale d'un système automatisé est illustrée à la figure I.2. Elle se compose de : partie opérative, partie commande, etc.



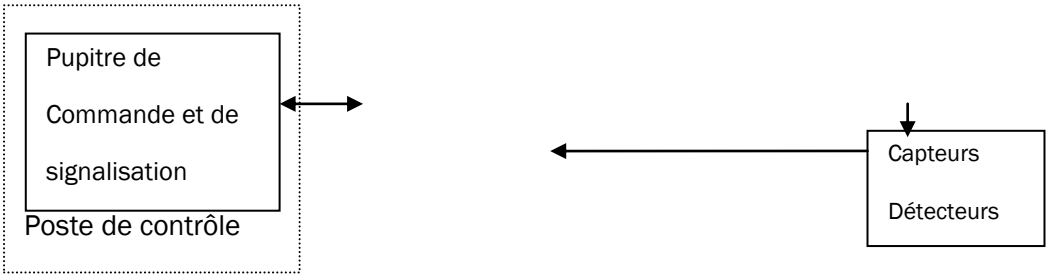


Figure I.2: un système automatisé

I.5.1 Partie opérative (machine)

La partie opérative est le processus physique à automatiser. Elle opère sur la matière et les produits entrants pour la transformation. Elle comporte en général de 3 types d'éléments : les capteurs et les actionneurs et les pré-actionneurs.

a) **Les Capteures :** Un capteur est un organe de prélèvement d'informations qui élabore à partir d'une grandeur physique, une autre grandeur physique de nature différente (souvent électrique). Les captures servant à détecter des positions physiques, des pressions, des températures, des forces, des vitesses, etc. L'information captée par la partie opérative est transmise à la partie commande par l'intermédiaire d'une interface d'entrée. On peut classer les capteurs en 2 groupes en fonction de la nature de l'information délivrée en sortie.

b) **Les actionneurs :** Un objet technique qui convertit une énergie d'entrée non directement utilisable par les mécanismes agissant sur la matière d'œuvre en une énergie de sortie utilisable pour obtenir une action définie.

c) **Les pré-actionneurs électriques :** Pour alimenter un moteur électrique, il est nécessaire de disposer d'un organe de commutation commandable électriquement par courant base tension appelé contacteur (lorsqu'il est commandé manuellement on l'appelle commutateur). Un relai est constitué d'une bobine alimentée par un circuit de commande dont le noyau mobile provoque la commutation de contacts pouvant être insérés dans un circuit de puissance. Un contacteur est un relai particulier pouvant commuter de fortes puissances grâce à un dispositif de coupure électrique. Actuellement sont réunis dans un seul boîtier un relais thermique, un contacteur, un disjoncteur... Les variateurs de vitesse font aussi partie des pré-actionneurs électriques.

I.5.2 Partie commande (automate) P.C

Elle donne les ordres de fonctionnement à la partie opérative. Donc elle :

• Emet les ordres de fonctionnement à la partie opérative. Ces ordres sont transmis aux Pré-actionneurs.

• Reçoit les informations transmises par les capteurs relatives à la situation de la partie Opérative. • Reçoit des consignes de fonctionnement en provenance de pupitre de commande.

• Émet les signaux de signalisation

• Assure le traitement des informations suivant une logique donnée (programme), afin d'élaborer les ordres.

I.6 Avantages et inconvénients d'un système automatisé :

I.6.1 Les avantages :

▪ La capacité de production accélérée

▪ L'aptitude à convenir à tous les milieux de production ;

▪ La souplesse d'utilisation.

▪ La création de postes d'automaticiens.

▪ Gain du temps.

I.6.2 les inconvénients :

▪ Le coût élève du matériel, principalement avec les systèmes hydrauliques.

▪ La maintenance doit être structurée.

▪ La suppression d'emplois.

Chapitre II

Généralités sur les API

II. Les automates programmables industriels :

II.1 Introduction

Dans le monde industriel, les exigences attendues de l'automatisation ont bien évolué. Parmi les éléments les plus répandus dans un système automatisé est API (l'automate programmable industriel)

. Ils sont apparus aux États-Unis vers la fin des années soixante, à la demande de l'industrie automobile américaine qui réclamait plus d'adaptabilité de leurs systèmes de commande. Les ingénieurs américains ont résolu le problème en créant un nouveau type de produit nommée automates programmables. L'API est la première machine à langage c'est-à-dire un des calculateurs logiques dont le jeu d'instruction est orienté vers les problèmes de logique et des systèmes a évolution Séquentielles.

Ils n'étaient rentables que pour des installations d'une certaine complexité, mais la situation a très vite changé, ce qui a rendu les systèmes câblés obsolètes. De nombreux modèles d'automates sont aujourd'hui disponibles ; depuis les nano automates bien adaptés aux machines et aux installations simples avec un petit nombre d'entrées/sorties, jusqu'aux automates multifonctions capables de gérer plusieurs entrées/sorties et destinés au pilotage de processus complexes

II.2 Logique câblée:

La technologie câblée consiste à raccorder des modules par des liaisons matérielles selon un schéma fourni par la description. Ces modules peuvent être électromagnétiques, électriques, pneumatiques ou fluidiques. En électricité ou en électronique, les liaisons sont faites par câble électrique. En pneumatique et fluidique, il s'agit de canalisations reliant les différents composants. Figure II.1 illustre une armoire électrique.

Figure II.1: Exemple d'une armoire électrique

Les outils câblés sont utilisés dans l'industrie où l'on apprécie ; à savoir la rapidité et le parallélisme. Ils souffrent cependant d'un certain nombre de limitations parmi lesquelles nous citons :

. • Leur encombrement (poids et volume).

• Leur manque de souplesse vis-à-vis de la mise au point des commandes et de l'évolution de celles-ci (améliorations, nouvelles fonctions, modification, etc.) : Toute modification impose la modification de câblage voire un changement de composants

. • Leur difficulté de maîtriser des problèmes complexes.

• La complexité de recherche des pannes et donc du dépannage.

• Leur coût élevé pour les systèmes complexes.

II.3 La logique programmée:

La technologie programmable consiste c'est-à-dire des machines destinées à traiter de l'information. Leur utilisation en gestion et en calcul scientifique est connue. Alors, les applications techniques relèvent de l'informatique à substituer le fonctionnement de

l'automatise par un programme chargé sur un constituant programmable industrielle. L'informatique industrielle est une discipline conjuguant les théories de l'automatique et les moyens de l'informatique dans le but de résoudre des problèmes de nature industrielle

Figure II.**2** : si dessous représente une armoire d'un API

Figure II.2: Exemple d'une armoire d'Api (logique programmée)

L'informatique offre donc une alternative technologique à l'automaticien et lui ouvre des possibilités nouvelles liées à la puissance de traitement et aux facilités de mémorisation de l'information. En termes d'avantage, nous citons :

• Moins de câble et d'encombrement.

• Fiabilité de l'automatisme.

• Facilité de modification

. • Flexibilité.

• Résolution des problèmes complexes. Le constituant programmable peut être soit un micro-ordinateur, soit une carte électronique ou bien un automate programmable.

II.4 Historique :

Au début des années 50, les ingénieurs étaient déjà confrontés à des problèmes d'automatismes, les composants de base de l'époque étaient les relais électromagnétiques à un ou plusieurs contacts. Les circuits conçus comportaient des centaines voire des milliers de relais. Le transistor n'était connu que comme un composant d'avenir et les circuits intégrés étaient inconnus.

Vers 1960, les semi-conducteurs (transistors, diodes) sont apparus dans les automatismes sous forme de circuits digitaux. Ce n'est que quelques années plus tard, que l'apparition des circuits intégrés a amorcé une révolution dans la façon de concevoir les automatismes. Ceux-ci étaient très peu encombrants et leur consommation était des plus réduite. On pouvait alors concevoir des fonctions de plus en plus complexes à des coûts toujours décroissants.

C'est en 1969 que le constructeur américain d'automobiles General Motors, a demandé aux firmes fournissant le matériel d'automatisme des systèmes plus évolués et plus souples pouvant être modifiés simplement sans coûts exorbitants.

Les ingénieurs américains ont résolu le problème en créant un nouveau type de produit nommé automates programmables. Ils n'étaient rentables que pour des installations d'une certaine complexité, mais la situation a très vite changé, ce qui a rendu les systèmes câblés obsolètes.

De nombreux modèles d'automates sont aujourd'hui disponibles ; depuis les nano automates bien adaptés aux machines et aux installations simples avec un petit nombre d'entrées/sorties, jusqu'aux automates multifonctions capables de gérer plusieurs milliers d'entrées/sorties et destinés au pilotage de processus complexes.

II.5 Définition d'un API :

Un Automate programmable Industriel (API) est une machine électronique

Programmable par un personnel automaticien (non informaticien) et destiné à piloter en

Ambiance industrielle et temps réel des procédés ou parties opératives.

Un automate programmable est adaptable à un maximum d'application, d'une point de Vue traitement, composants, langages, c'est pour cela qu'il est de construction modulaire. On cite par exemple des instructions composantes les fonctions d'automatisme:

- Logique séquentielle et combinatoire ;
- Temporisation, comptage, décomptage, comparaison ;
- Calcul arithmétique ;

- Réglage, asservissement, régulation, etc. …, pour commander, mesurer et contrôler au moyen d'entrées et sorties (logiques, numériques ou analogiques) différentes sortes de machines Ou de processus, en environnement industriel.

II.6 Les Avantages des API

L'utilisation de l'automate programmable dans le domaine industrielle présenté plusieurs avantages, dans la suite on va illustrer les plus importantes.

▪ Moins de constituants La substitution des relais à un gain en volume, on encombrement et à la simplicité de l'emploi, particulièrement apprécies sur les machines simples.

▪ Moins de câblage Les connections se réduisent au raccordement des capteurs aux entrées et des prés actionneurs aux sorties. L'accès aux différents organes de l'automatisme, lots des modifications et des réglages, se trouve ainsi facile.

▪ Plus de confort Le programme qui se substitue au câblage et l'ensemble des graphiques on peut le saisir, le modifier et l'archiver facilement grâce au terminal de programmation et de réglage. Ce programme peut être duplique pour les machines construites ainsi une diminution des coûts.

▪ Plus de fonctionnalités Pour les machines spéciales où leurs installations sont compliquées, l'automate programmable offre des fonctions d'automatisme spécifiquement intégrées.

▪ Plus d'information La maintenance et la mise en place d'un automatisme est facile par la visualisation permanente de l'état des entrées/sorties, qui sont signalés par des voyants lumineux. Le dialogue entre l'homme et la machine est assuré par un terminal de programmation ainsi de réglage grâce à son mode conversationnel et les messages affichés sur l'écran

II.7 Architecture d'un automate

II.7.1 Structure externe (figure II.3):

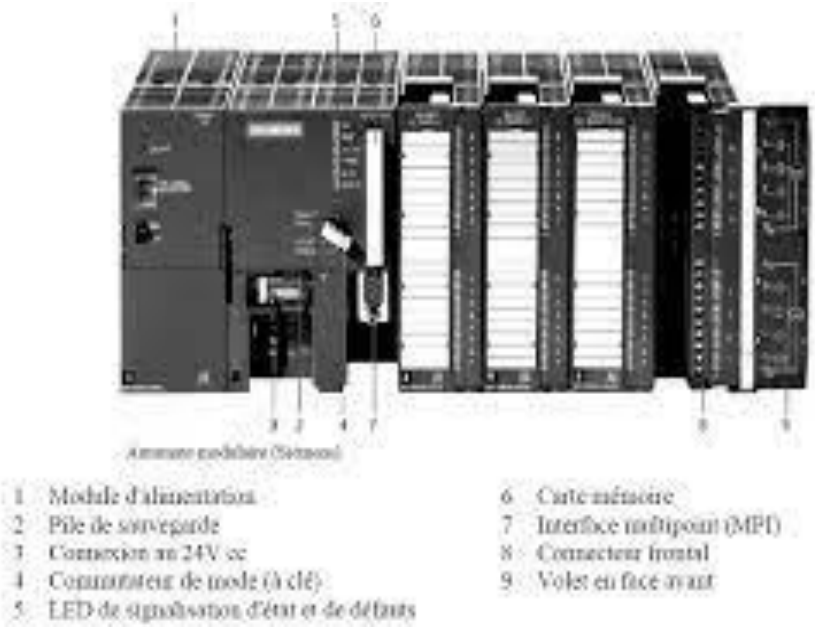


Figure II.3 : structure externe d'un API

II.7.2 Structure interne des automates programmables

. Un automate programmable industriel est donc constitué de :

• Un processeur.

• Une mémoire.

• Des interfaces d'Entrées/Sorties.

• Une alimentation (240 V → 24 VPC). Ces quatre parties sont reliées entre elles par des bus (ensemble câble autorisant le passage de l'information entre ces 4 secteurs de l'API).

Figure II.4 illustre la structure interne d'un API.

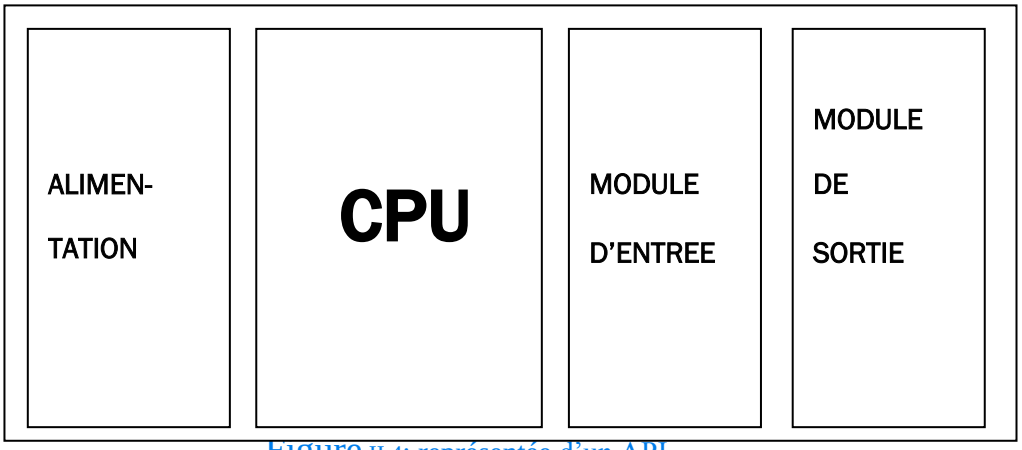


Figure II.4: représentée d'un API

II.7.2.1 L'unité centrale de traitement (CPU) : Central Procession Unit)

C'est le cœur de la machine, Processeur appelé unité de traitement, il assure le contrôle de l'ensemble de la machine et effectue les traitements demandés par les instructions du programme. Il réalise les fonctions logiques, temporisation, comptage, calcul. Il comporte un certain nombre de registres (compteur ordinal, registre d'instructions, registre d'adresse, registres de données, accumulateurs, ... Il est connecté aux autres éléments (mémoires, interfaces d'E/S, ...) par l'intermédiaire des bus. Figure II.5 montre le CPU de l'automate.

Figure II.5 : exemple d'un CPU (CPU 313)

II.7.2.2 Mémoires :

La mémoire centrale est découpée en plusieurs zones :

• Zone mémoire programme ;

• Zone mémoire des données (états des E/S, valeurs des compteur, temporisations, ...) ;

• Zone où sont stockés des résultats de calcul utilisé ultérieurement dans le programme ;

• Zone pour les variables internes. Il existe différents types de mémoires :

a)Mémoires vives :

RAM ce sont des mémoires volatiles : est utilisée pour stocker les données et les programmes lors du fonctionnement ; elles perdent l'information en cas de coupure de

l'alimentation. Certaines d'elles sont équipées de batteries de sauvegarde (autonomie réduite). Elles sont accessibles en lecture et en écriture.

b) Mémoires mortes (ROM) :

Les contenus sont figés. Ce sont des mémoires à lecture seule. Les informations sont conservées en permanence sans source externe

Mémoire programmé par le fabriquant et ineffaçables ; elle représente un espace de stockage permanent pour le système d'exploitation et les données figées utilisées par le CPU (PROM, REPROM ou EPROM, EEPROM).

II.7.2.3 Un module d'alimentation :

Il assure la distribution d'énergie aux différents modules. Partir d'une tension 220 v ,50HZ ou dans certains cas de 24V fournit les tensions continues +/-5V, +/-12V ou +/-15V nécessaire ou bon fonctionnement du système.

II.7.2.4 Un ou plusieurs modules d'entrées : tout ou rien ou analogiques

pour l'acquisition des informations provenant de la partie opérative (procédé à conduire).

II.7.2.5 un ou plusieurs modules de sorties : 'tout ou rien ' ou analogiques

pour transmettre à la partie opérative les signaux de commande.

Remarquons qu'il existe des modules qui intègrent en même temps des entrées et sorties.

Un ou plusieurs modules de communication (ou coupleurs) comprenant :

-Interface(s) série utilisant dans la plupart des cas comme support de communication

Les liaisons RS-232 OU RS422/RS485 pour assurer la connexion à des terminaux (console, ou PC)

Pour assurer la communication homme/machine (programmation, supervision ….).

II.7.2.6 Les bus :

Le bus (interne) est un ensemble de pistes conductrices (pistes en cuivre) par lequel s'achemine une information binaire (suite de 0 ou 1) sur chaque fil :

▪ Le bus de données transporte les données utilisées dans les traitements effectués par le CPU.

▪ Le bus d'adresses transporte les adresses des emplacements mémoire que le CPU utilise pour accéder aux données enregistrées dans ces emplacements. ▪ Le bus de commandes transporte les signaux utilisés par le CPU pour le contrôle, tels que tops de synchronisation, sens des échanges, contrôle de validité des échanges, etc.

. Figure II.6 montre Les différents types de bus

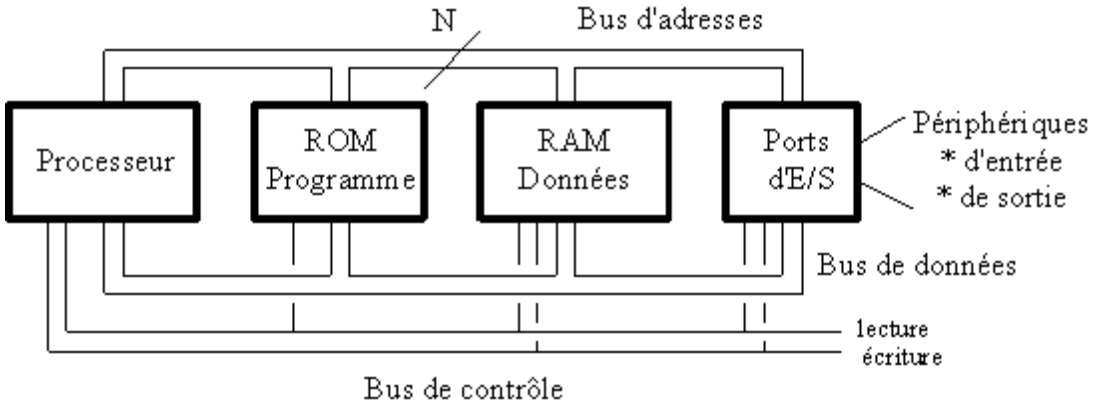


Figure II.6 : Les différents types de bus

II.8 Critères de choix d'un automate

Le choix d'un automate programmable est en premier lieu le choix d'une société ou d'un Groupe et les contacts commerciaux et expériences vécues sont déjà un point de départ. Les grandes sociétés privilégieront deux fabricants pour faire jouer la concurrence et pouvoir Se retourné en cas de perte de vitesse de l'une d'entre elles. Le personnel de maintenance doit toutefois être formé sur ces matériels et un trop grand Diversité des matériels peut avoir de graves répercussions. Un automate utilisant des langages de programmation de type GRAFCET est également Préférable pour assurer les mises au point et dépannages dans les meilleures conditions. La possession d'un logiciel de programmation est aussi source d'économies (achat du logiciel Et formation du personnel). Des outils permettant une simulation des programmes sont Egalement souhaitables.

Il faut ensuite quantifier les besoins :

• Nombre d'entrées / sorties

• Type de processeur

• Fonctions ou modules spéciaux

• Fonctions de communication

II.9 La programmation de l'automate :

La programmation de l'automate est le rôle d'un automaticien. Le langage assembleur qui nécessite la maitrise de l'architecture interne de l'automate n'est pas recommandée.

Les langages évolués orientés Object (tels que le langage fortran, Turbo C, Pascal) et qui nécessitent des connaissances informatique poussés sont très peu utilisés.

Pour ces raisons, des langages spéciaux, dont les instructions sont, souvent, représentés par des symboles proches de ceux utilisés en automatismes.

Actuellement, presque la totalité des fabricants d'automates utilisent les langages standards définis dans la norme IEC 61131-3.

Ces langages se résument en :

Trois langages graphiques :

-LD : Langage à contacts **(CONT)** ou **LADDER (LD)**

-FBD : langage en blocs fonctionnels

-SFC : (séquentiel fonction char) ou GRAFCET. Ces deux langages sont très proches.

- ✓ **Deux langages littéraux :**

-IL : liste d'instructions.

-ST : littéral structuré (très proche du langage **Pascal** et du **C)**

II.10 Conclusion

Dans ce chapitre, on a donné un aperçu sur les API et leur fonction dans un milieu industrielle ; la nature des informations traitées ; ainsi que la structure interne et externe des API et on propose les points essentiels pour choisir un API qui réponde à des besoins bien spécifiques

Chapitre III

Le Language ladder

III.1.Introduction :

un langage de programmation agit comme un canal de communication qui permet aux êtres humains d'interagir avec les machines en leur donnant des instructions et en analysant les informations matérielles fournies par le système, généralement un ordinateur .Ce langage permet au rédacteur d'un programme de se distancer des mécanismes internes, tels que les activations et les désactivations de commutateurs électroniques, qui conduisent au résultats souhaité. L'acte de rédaction du code source d'un programme est appelé programmation.

Il implique l'utilisation de méthodes pour écrire et résoudre des algorithmes informatique, qui reposent sur des principes logiques.

Un langage de programmation se différencie d'un langage mathématique par sa vocation pratique (une fonction doit, par extension, retourner une valeur).en conséquence, un « langage de programmation est toujours un compromis entre la puissance et la possibilité d'exécution.

III.2.définition :

Ladder diagramme (LD) ou langage ladder ou schéma à contacts est une langage graphique très populaire auprès des automaticiens pour programmer les automates programmable industriels .il ressemble un peu aux schémas électriques, en plus il est facilement compréhensible .ladder est le mot anglais pour échelle, figure représente un exemple de programme avec le langage ladder.

III.3.Origine :

L'idée initiale du Ladder est la représentation de fonction logique sous la forme de schémas électriques. Cette représentation est originellement matérielle : quand l'automate programmable industriel n'existait pas, les fonctions étaient réalisées par des câblages. Par exemple, pour réaliser un ET logique avec des interrupteurs, il suffit de les mettre en série. Pour réaliser un OU logique, il faut les mettre en parallèle.

Le Ladder a été créé et normalisé dans la norme CEI 61131-3. Il est encore aujourd'hui souvent utilisé dans la programmation des automates programmables industriels, bien qu'ayant tendance à être délaissé en faveur de langages plus évolués, et plus adaptés aux techniques modernes de programmation, tels que le ST par exemple, ou encore le **Grafcet**, plus adapté à la programmation de séquences.

III.4.Principe :

Un programme Ladder se lit de haut en bas et l'évaluation des valeurs se fait de gauche à droite. Les valeurs correspondent en fait, si on le compare à un schéma électrique, à la présence ou non d'un potentiel électrique à chaque nœud de connexion.

En effet, le Ladder est basé sur le principe d'une alimentation en tension représentée par deux traits verticaux reliés horizontalement par des bobines, des contacts et des blocs fonctionnels, d'où le nom 'Ladder' (échelle).

C'est un langage volontairement simple et graphique pour être compréhensible. Cela a permis, dans les années 1990, son utilisation sans formation lourde par les électriciens. Il est aujourd'hui un peu dépassé.
La puissance de calcul des CPU actuelles permettent de travailler directement en langage objets avec des notions de classe et d'héritage.

III.5.Les composants du langage LADDER :

Il existe 3 types d'élément de langage :

- les entrées (ou contact), qui permettent de lire la valeur d'une variable booléenne ;
- les sorties (ou bobines) qui permettent d'écrire la valeur d'une variable booléenne ;
- les blocs fonctionnels qui permettent de réaliser des fonctions avancées

les composants graphiques élémentaires d'un diagramme LD sont :

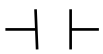

Barre d'alimentation à droite

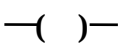

Barre d'alimentation à gauche

Arc de liaison horizontal

Arc de liaison vertical

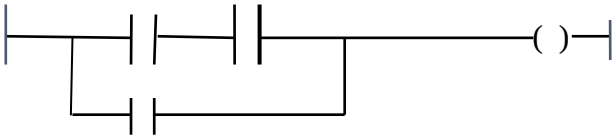

Liaison multiples verticales et horizontales


Contact associé à une variable

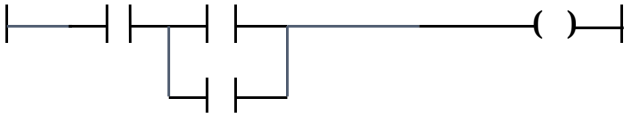

Relais associé à une variable de sortie ou interne

III.5.1.Barre d'alimentation :

Un diagramme LD est limité sur la gauche et la droite par des lignes verticales appelées respectivement barre d'alimentation à gauche et barre d'alimentation à droite .



Les symboles du diagramme LD sont reliés entre eux et aux barres d'alimentation par Des **arcs de liaisons** verticaux ou horizontaux.



Chaque segment de liaison peut prendre l'état booléen FALSE ou TRUE. L'état d'une Liaison est propagée dans toutes ses extrémités à droite. Toute liaison horizontale connectée à la barre d'alimentation à gauche a l'état booléen TRUE.

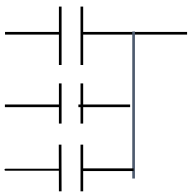
III.5.2.liaison multiple :

La combinaison de liaison horizontale et verticale permet la construction de liaisons Multiples. L'état aux extrémités d'une liaison multiple respecte des règles logiques.

Une liaison multiple à gauche combine plusieurs liaisons horizontales connectées à Gauche d'une liaison verticale, et une seule liaison horizontale à droite.

L'état de l'extrémité à droite est le résultat du OU logique entre les états des extrémités à gauche.

Exemple de liaison multiple à gauche : de l'extrémité droite (V1or V2 or V3) .



Une liaison multiple à droite combine une liaison horizontale connectée à gauche d'une liaison verticale , et plusieurs liaisons horizontale à droite .

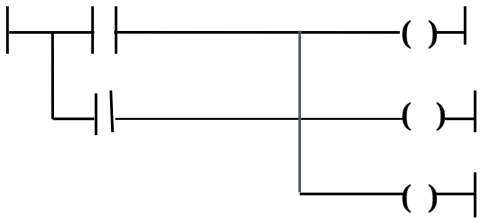
l'état de l'extrémité à gauche est propagé dans toutes les extrémités à droite .

Exemple de liaison multiple à droite :

Output1 := input ; uotput2 := input2.

Une liaison multiple à gauche et à droite combine plusieurs liaisons horizontales Connectées à gauche d'une liaison verticale, et plusieurs liaisons horizontales à droite. L'état De chacune des extrémités à droite est le résultat du UO **logique** entre les états des extrémités à gauche.

Exemple de liaison multiple à gauche et à droite : output1 OR input2 ; output2 := Input1 OR input2; output3:= input1 OR input2 ;



II.5.3.contacts et relais :

Les type des contacts suivent peuvent être utilisés dans un diagramme :

-contact direct ;

-contact inversé ;

-contacts à détection de front.

Les types de relais suivent peuvent être utilisés dans un diagramme :

-relais direct ;

-relais inversé ;

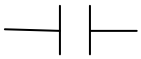
-relais à forçage de type SET ;

-relais -relais à forçage de type RESET ;

-relais à détection de front.

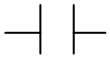
Le nom de la variable associée est inscrit au-dessus du symbole graphique :

Bool_variable



a)-Contact direct :

Un contact direct représente une opération entre l'état d'un arc de liaison et une variable booléenne. Bool_variable



L'état de l'liaison à droite est le résultat du **ET logique** entre l'état de la liaison à gauche et la valeur de la variable associée au contact.

b)-Contact inversé :

Un contact inversé représente une opération entre l'état d'un arc de liaison et l'inverse logique d'une variable booléenne. Bool_variable

c)-contact à détection de front positif :

Un tel contact représente une opération entre l'état d'un arc de liaison et le passage à l'état TRUE d'une variable booléenne : front montant.

L'état de liaison à droite prend l'état TRUE quand l'état de la liaison à gauche est à TRUE et que l'état de la variable associée passe de FALSE à TRUE .il prend l'état FALSE dans tous les autres cas .

d)-Contact à détection de front négatif :

Un tel contact représente une opération une entre l'état d'un arc de liaison et le passage à l'état FALSE d'une variable booléenne :

L'état de la liaison a droite prend l'état TRUE quand l'état de la liaison à gauche est à TRUE et que l'état de la variable associée passe de TRUE à FALSE .IL prend l'état FALSE dans tous les autres cas.

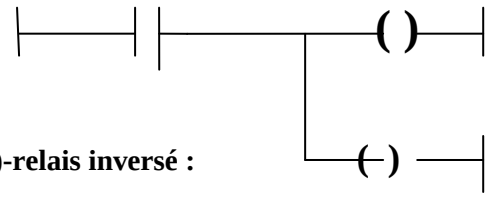
e)-relais direct : UN relais direct permet le forçage d'une sortie booléenne avec l'état d'un arc de liaison.

La variable associée est **forcée à l'état de la liaison à gauche.**

l'état de la liaison à gauche est propagé dans la liaison à droite .la liaison à droite peut être connectée à la barre d'alimentation à droite .la variable booléenne associée doit être de sortie ou interne.

le nom de la variable associée peut être le nom du programme édité (pour les fonctions seulement).dans ce cas, le relais représente l'affection de la valeur renvoyée par la fonction.

Exemple de relais directs :output1 :=input2 :=input1 ;



f)-relais inversé :

Un relais inversé permet de forçage d'une sortie booléenne avec l'inverse logique de l'état d'un arc de liaison. Bool_variabl

—(/)—

La variable associée est forcée à l'inverse logique de l'état de la liaison à gauche .l'état

De la liaison à gauche est propagé dans la liaison à droite. la liaison à droite peut être

Connectée à la barre d'alimentation à droite.

La variable booléenne associée doit être de sortie ou internet. Le nom de la variable

Associée peut être le nom du programme édité (pour les fonctions seulement). Dans ce cas , le relais représente l'affection de la valeur renvoyée par la fonction.

la variable associée est forcée à l'inverse logique de l'état de la liaison à gauche. L'état

De la liaison à gauche est propagé dans la liaison à droite. La liaison à droite peut être

Connectée à la barre d'alimentation à droite. La variable booléenne associée doit être de sortie

Ou interne.

le nom de la variable associée peut être le nom du programme édité (pour les fonctions seulement). Dans ce cas, le relais représente l'affection de la valeur renvoyée par la fonction.

g)-relais à action SET :

un relais à action **SET** permet le forçage d'une sortie booléenne **en fonction de l'état d'un arc de liaison.**

Bool_variable

—(S)—

La variable associée **est forcée à TRUE** quand la liaison à gauche prend l'état TRUE.

La variable garde cette valeur jusqu'à un ordre inverse donné par un relais de type RESET.

L'état de la liaison à gauche est propagé dans la liaison à droite. la liaison à droite peut être

Connectée à la barre d'alimentation à droite. la variable booléenne associée doit être de sortie ou internet.

h)-Relais à détection de front montant :

un relais positif permet le forçage d'une sotie booléenne en fonction de l'état d'un arc

de liaison. Bool_variable

——(p)——

La variable est **forcée à TRUE** quand la liaison à gauche passe de l'état FALSE à l'état

TRUE : **front montant**. la variable est remise à FALSE dans tout les autres cas. L'état de la

Liaison à gauche est propagée dans la liaison à droite. la liaison à droite peut être connectée à

La barre d'alimentation à droite. La variable booléenne associée doit être de sortie ou interne.

i)-relais à détection de front descendant :

un relais négatif permet le forçage d'une sortie booléenne en fonction de l'état d'un arc de liaison.

Bool-variable

—(N)—

la variable est **forcée à TRUE** quand la liaison à gauche passe de l'état TRUE à l'état

FALSE : **front descendant** .la variable est remise à FALSE dans tous les autres cas .l'état de la liaison à gauche est propagé dans la liaison à droite .la liaison à droite peut être connecté à la barre d'alimentation à droite .la variable booléenne associée doit être de sortie ou interne.

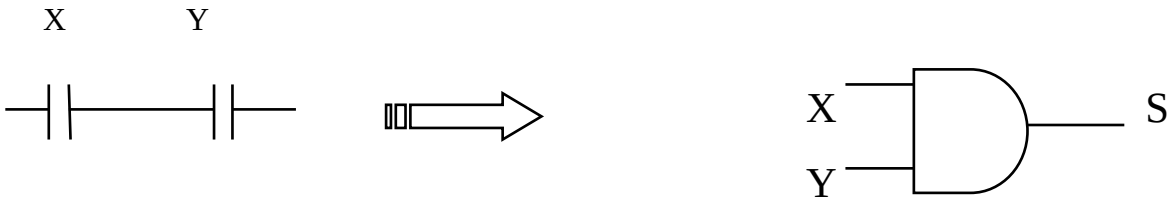
Le langage LD n'est qu'une représentation graphique des fonctions booléennes fondées sur une analogie avec les relais électromécaniques. Dans un programme LD, il est difficile de détecter des erreurs ou de faire la mise au point .car sa représentation graphique cache le flux

séquentiel de contrôle implicite dans un programme .Ceci est avantage évident dans un langage comme SFC.

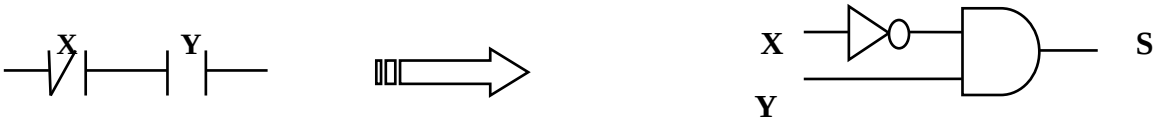
III.6. Réalisation de fonction logique :

Comme dit précédemment, les fonctions logiques sont dérivées de leurs réalisations électriques. Donc chaque fonction logique (AND, OR, XOR, NAND, NOR, NOT) a une représentation qui correspond à son équivalent électrique.

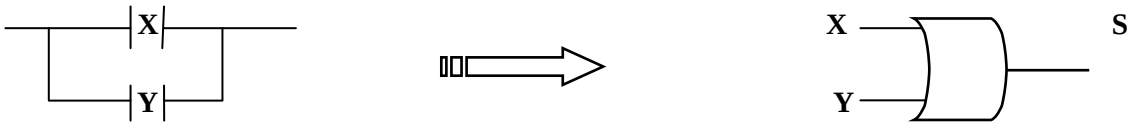
C'est-à-dire :



Equivaut à X AND Y → (X*Y)

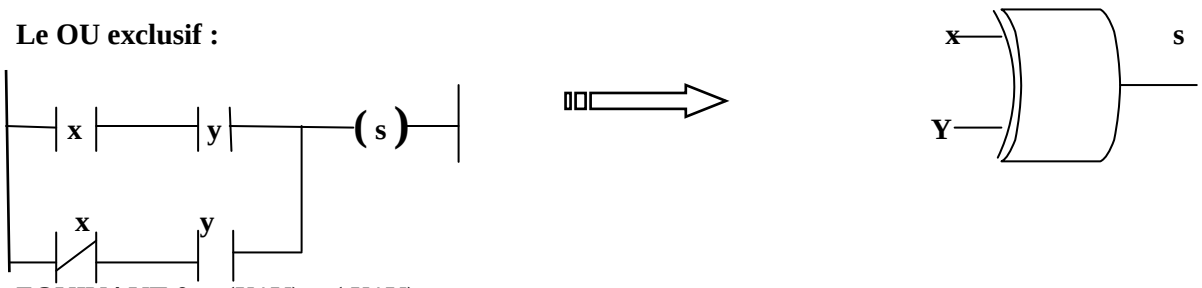


Équivaut à NOT(X) AND Y



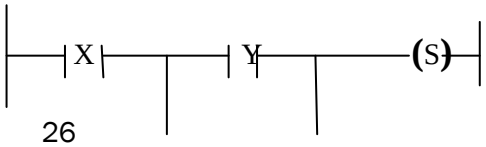
Équivaut à X OR Y → (x+y)

Le OU exclusif :

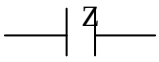


EQUIVAUT 0 s=(X*Y) + (X*Y)

Plus complexe :



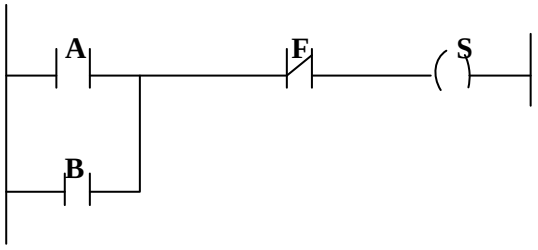
équivaut à S = X.(Y+Z)



III.7.La lecture :

Un programme LADDER se lit de haut en bas et l'évaluation des valeurs se fait de gauche à droite. Les valeurs correspondent en fait, si on le compare à un schéma électrique, à la présence ou non d'un potentiel électrique à chaque nœud de connexion.

Exemple de lecture



Dans ce réseau, si A OU B est actionnée ET si F n'est pas actionné, la sortie S est active; soit

S = (A + B). (/F) ; Le signe "/F" signifie l'inversion de l'entrée "F", cela se prononce "F barre".

Chapitre IV

L'environnement Delphi et le language pascal

IV. L'environnement Delphi et le langage pascal.

IV.1.Delphi :

IV.1.1. Introduction :

Avant l'émergence de la programmation visuelle, les développeurs devaient investir considérablement de temps à rédiger du code étape par étape, en prenant grand soin de chaque détail. Identifier une erreur de syntaxe prenait énormément de temps, car il n'existait aucun moyen de la repérer automatiquement

.L'avènement de langages tels que Delphi, Visual Basic, C++Builder... a grandement facilité la tâche, grâce aux environnements de développement intégrés (EDI). Ces environnements offrent la possibilité de créer des applications complètes sans nécessiter l'écriture manuelle de chaque ligne de code. Comme de nombreux logiciels, Delphi existe sous différentes versions (Delphi 1,..., Delphi 7, Delphi 2006,..., Delphi 2010, Delphi XE).

Pour ce manuel en particulier, nous nous focalisons sur la version 7 ,. Ce chapitre initie les apprenants à l'environnement de développement intégré propre à Delphi, leur permettant ainsi de se familiariser avec son fonctionnement.

IV.1.2. définition de Delphi :

Borland Delphi se présente comme un environnement de programmation à orientation objet, destiné à la conception d'applications 32 bits pour les systèmes d'exploitation Windows et Linux. En optant pour Delphi, il devient possible d'élaborer des applications de grande envergure en limitant la nécessité de s'engager dans une programmation approfondie. Delphi tire parti du langage Pascal pour ses réalisations. Pascal, un langage de programmation sophistiqué, sert de base à cette plateforme. Delphi se présente comme un logiciel conçu pour la création d'applications en utilisant ce langage. Il est important de noter que Delphi représente davantage qu'une simple couche superficielle, agissant comme une enveloppe confortable autour de Pascal. En simplifiant de nombreuses tâches liées à la programmation en Pascal, Delphi facilite le processus. Il convient également de souligner que Delphi est

spécifiquement orienté vers le développement de programmes compatibles exclusivement avec l'environnement Windows.

IV.1.3. Historique :

Anders Hejlsberg, le créateur de Turbo Pascal, a joué un rôle essentiel en tant que l'un des chefs de projet chez Borland pour le développement de Delphi. Par la suite, lorsqu'il a rejoint Microsoft, Hejlsberg est devenu le cerveau derrière la création du langage C#. La version actuelle de Delphi, qui est la version 2007, regroupe au sein d'un environnement de développement intégré (EDI) la capacité de produire du code pour diverses cibles telles que Win32, Delphi, .NET, ainsi que C#.L'adaptation du langage Delphi pour .NET s'est faite afin de correspondre à la structure de la plateforme .NET de Microsoft, tandis que des améliorations ont été apportées au langage Delphi pour Win32. En précisant, Delphi 7 est conçu pour le développement sur l'environnement Win32 de Microsoft, tandis que Delphi 8 est axé sur l'environnement .NET de Microsoft. Ces deux versions utilisent des compilateurs distincts. La fusion entre les aspects Win32 et .NET n'a été accomplie qu'à partir de Delphi 2005. De même, la bibliothèque de composants visuels (VCL) a été ajustée pour Delphi 8 et Delphi 2005, dans le but de simplifier la migration des applications Delphi Win32 vers .NET. À titre d'exemple, toutes les mentions de pointeurs ont été substituées par des références vers des objets.

IV.1.4. Un peu de vocabulaire :

« POO » programmation orienté objet :

C'est une façon de développer et de présenter une application informatique sous la forme d'objets, ayant des propriétés.

«**Programme** » : texte écrit dans un langage informatique, comportant dans notre cas des instructions structurées. Il est destiné à être « converti » par Delphi en un logiciel utilisable sous Windows.
« **Développe*r*** en Delphi » : écrire des programmes en utilisant Pascal.
« **Application** » : Logiciel fonctionnant sous Windows.
«**Projet** » : c'est la base d'une application. Sous Delphi, pour créer une application, on constitue d'abord un projet.

<<Code » : (Code Pascal, Code source) : morceau de programme, texte d'un programme écrit en Pascal.

« **Interface** (utilisateur) » : la partie d'un logiciel qui est visible par l'utilisateur, à l'opposé du code source, invisible à l'utilisateur.

« **Fiche** » : fenêtre à l'état non compilé. Les fiches sous Delphi ou fenêtres sous Windows.

IV.1.5. Description de l'environnement de développement intégré Delphi (EDI) :

On appel EDI « Environnement de Développement Intégré » , L'interface proposée par Delphi pour assister les utilisateurs dans la création de leurs applications rappelle un environnement semblable à un atelier virtuel. Dans cet espace, une boîte à outils ainsi qu'une variété d'éléments sont mis à disposition des utilisateurs. Démarrer Delphi peut être réalisé selon différentes méthodes :

- ✓ Double cliquer sur l'icône Delphi (si vous avez créez un raccourci).
- ✓ Choisir Tous les Programmes/Borland Delphi 7/Delphi 7 dans le menu Démarrer de Windows.
- ✓ Choisir Exécuter dans le menu Démarrer puis entrer Delphi 32.
- ✓ Double cliquer sur Delphi32.exe dans le répertoire Delphi/Bin.

Au lancement du logiciel l'écran a l'aspect ci-dessous :

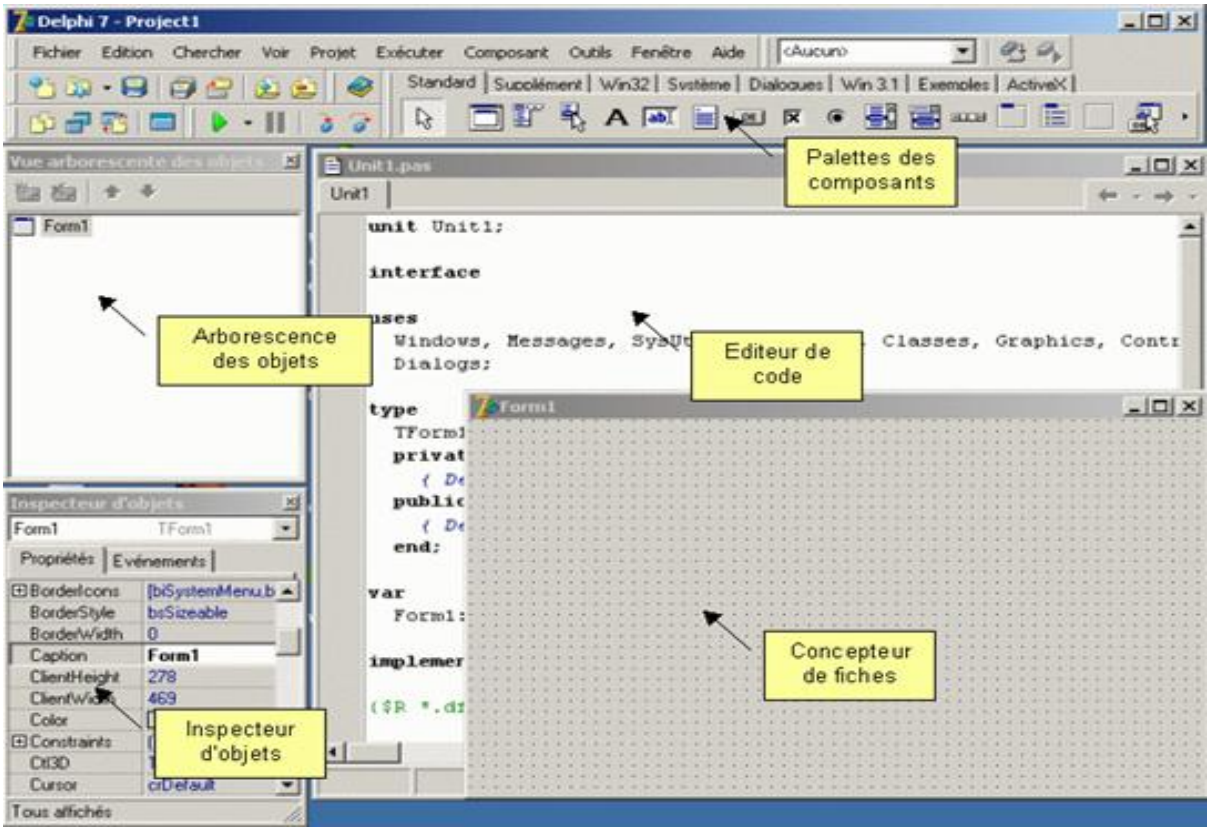


Figure IV 01 : la fenêtre Delphi 7

IV.1.5.1 La barre des menus :

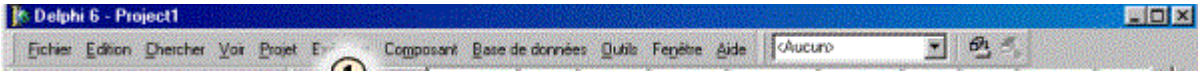


Figure IV 02 : Barre de menus

Delphi présente des menus comme toutes applications Windows. Vous ne trouverez aucune ambiguïté quant à leur utilisation.

La **barre de menus** permet d'accéder à la plupart des commandes disponibles.(voire annexe)

- ✓ Le menu **Fichier** permet d'ouvrir un nouveau fichier, un nouveau projet, d'enregistrer votre travail et d'imprimer.
- ✓ Le menu **Edition** donne accès aux fonctions copier coller classiques ainsi qu'a des outils de présentation,
- ✓ le menu **Recherche** permet d'effectuer des recherches dans de longs programmes,
- ✓ le menu **Voir** permet d'avoir accès aux différentes fenêtres de Delphi, d'afficher des éléments constituant une application,
- ✓ le menu **Projet** permet d'accéder aux commandes spécifiques au projet en cours.

- ✓ le menu **Exécuter** permet la compilation et l'exécution d'un programme.
- ✓ le menu **Outils** donne accès à divers outils de Delphi, donc un seul vraiment intéressant : l'éditeur d'images.
- ✓ Le menu **Aide** enfin, permet d'accéder à l'aide du logiciel.

- ➢ ne s'agit pas ici de connaître chacun des éléments de menus de Delphi, c'est inutile à ce stade car certaines commandes ne servent qu'en de rares occasions.
- ➢ Il est cependant essentiel de vous familiariser avec les commandes les plu s utiles pour la gestion des projets : Nouveau…, Ouvrir, Enregistrer, Enregistrer sous…, Fermer.(voire annexe)

IV.1.5.2 La barre d'outils : Sur le côté gauche de la fenêtre principale , vous découvrirez les ensembles de commandes. Ces ensembles de commandes au sein de Delphi offrent une manière pratique d'accéder rapidement aux fonctions et actions les plus fréquemment employées. Bon nombre des opérations accessibles via les ensembles de commandes sont également présentes dans les menus déroulants.

Figures 3 représente quelques Barres d'outils :

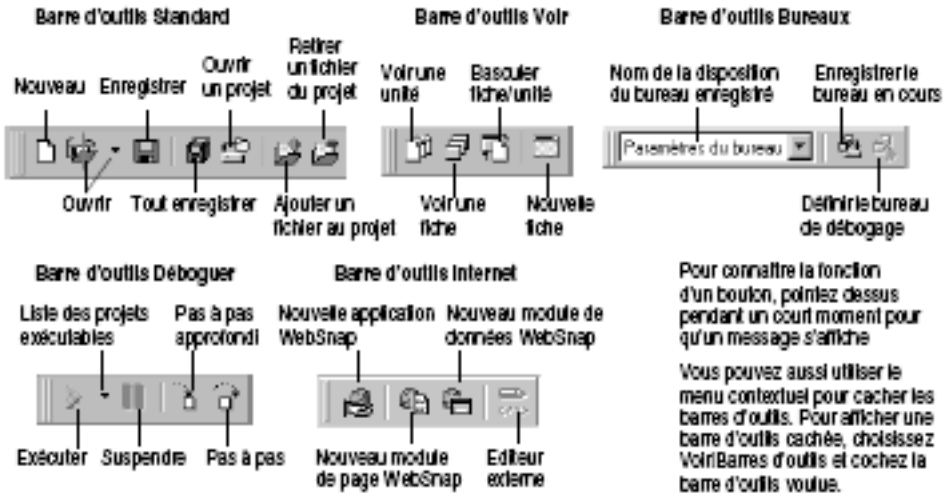


Figure IV 03 quelques Barres d'outils

IV.1.5.3. La palette des composants :

Cette palette , généralement positionnée vers la droite en dessous de menu (Figure 04), offre la possibilité d'explorer l'ensemble des composants utilisables dans Delphi. Ces composants (voire annexe) sont regroupés dans un classeur à onglets. Chaque onglet permet d'accéder à une variété de composants.

C'est là que nous découvrons les éléments fournis par Windows pour construire notre interface : boutons, listes, cases à cocher, arbres et listes, grilles, etc.

Figure 04 : La palette des composants

IV.1.5.4. La fiche et l'unité :

a) La fiche ou « forme »

est la fenêtre que vous allez voir lorsque vous lancez le programme.

Pour l'instant, elle est vide, mais c'est là que vous pouvez rajouter les objets (les menus, les boutons …etc.), vous pouvez l'agrandir, la réduire, bref, faire tout ce que vous voulez sans taper une ligne de code...

Une application peut comporter plusieurs fiches.

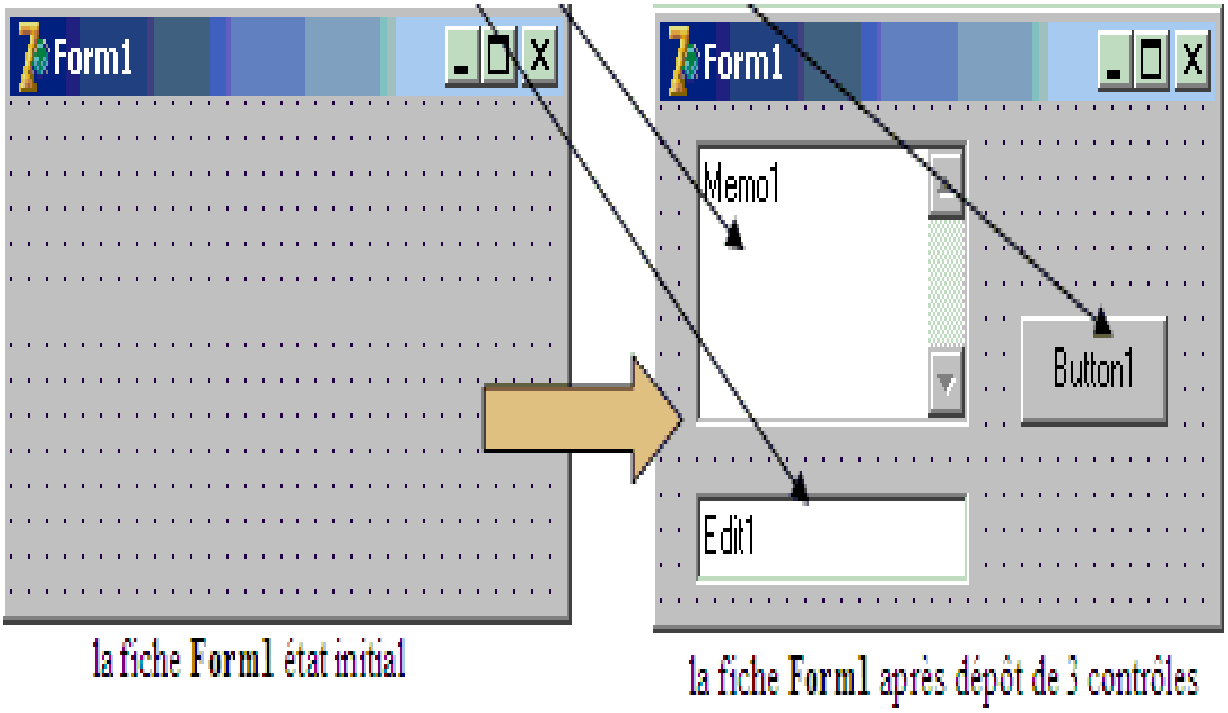


Figure 05 :exemple de fiche

b) L'unité : L'éditeur de code ou « unité » est généralement caché sous le concepteur de fiche, cette fenêtre contient le code de votre programme (fig.).

- Appuyer sur la touche **F12** pour l'amener en premier plan.
- Comme vous le voyer, il y a déjà du code écrit alors que vous n'y êtes pour rien .
- A ce propos , la relation entre une fiche et son unité est non modifiable .

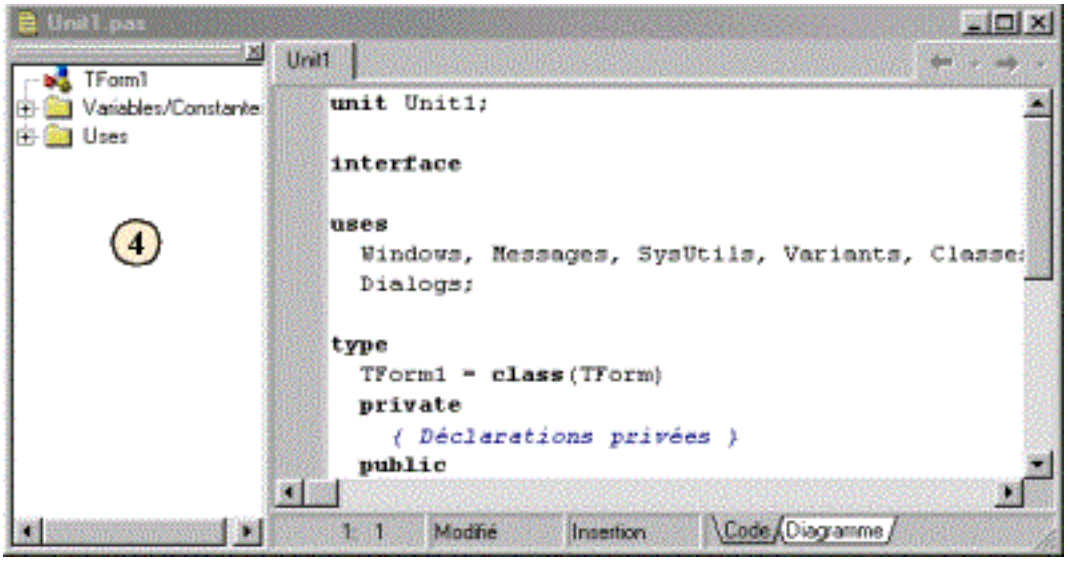


Figure 06 :exemple d'unité

Attention cependant à ne pas tomber dans l'excès : certaines unités n'auront absolument pas besoin de fiche correspondante (des unités comportant exclusivement des calculs mathématiques par exemple)

IV.1.5.5 L'inspecteur d'objet :

L'inspecteur d'objets de Delphi (**fig 07**) affiche l'ensemble des propriétés et l'ensemble des événements associés à un composant.

S'il n'est pas visible, appuyer sur la touche **F11** pour le faire apparaître.

- L'inspecteur d'objets vous permet de modifier les propriétés et d'accéder aux événements des composants et des fiches.
- L'inspecteur d'objets comporte deux pages :

Page propriétés qui vous permet, en mode conception, de définir les propriétés des composants de votre fiche, mais aussi celles de la fiche.

Page Evénements : qui vous permet d'associer des événements aux objets de votre projet.

Ici , on peut voir qu'il s'agit des caractéristique générales de la fenêtre car on voit **'' Forme 1 :TForme1''**

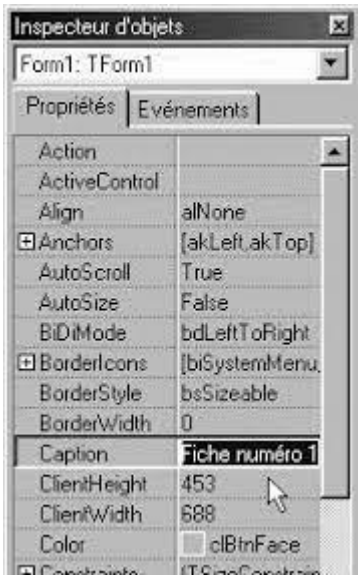


Figure 07 : L'inspecteur d'objets

IV.2. Pascal objet :

IV.2.1 Historique :

Pascal Objet (ou Object Pascal en anglais) est un langage orienté objet dérivé du **Pascal**. Il a été créé en 1990 par la société Borland comme une amélioration de son logiciel phare de l'époque, Turbo Pascal. Il s'agissait alors de la version 5.5. Même si l'ajout de l'objet au Turbo Pascal s'est opéré en douceur et a été vendu par Borland comme une simple et logique amélioration de Turbo Pascal, cela n'en a pas moins révolutionné ce langage et la communauté de développeurs qui lui était associée.

Le **Pascal Objet** prend un nouvel essor en 1995 avec la sortie de **Delphi** 1, toujours à l'initiative de Borland.

IV.2.2 Les données élémentaires dans un programme pascal :

Tout programme écrit en PASCAL implique la manipulation de données. Les types de données sont des étiquettes désignant des catégories spécifiques de données. Ces types permettent au compilateur PASCAL de comprendre comment stocker les données en

mémoire, quelle quantité de mémoire allouer, quelles valeurs sont valides, et quelles opérations peuvent être effectuées sur ces données.

Il est impératif d'attribuer un type spécifique à chaque donnée dans un programme, à quelques exceptions près qui sont généralement utilisées pour masquer des erreurs de conception dans le code.

1. Nombres :

Nous met à disposition diverses catégories de valeurs numériques, qu'elles soient des entiers ou des nombres décimaux, qui s'avèrent utiles dans de nombreuses applications.

2. Opérations Numériques :

Les valeurs numériques, qu'elles soient des entiers ou des nombres décimaux, sont utilisées pour effectuer des opérations mathématiques courantes telles que l'addition, la soustraction, la multiplication et la division. Ces opérations sont représentées par les symboles respectifs : + - * /. En langage Pascal, les opérations sont écrites de manière similaire à des calculs manuscrits, sans que les espaces entre les nombres et les signes d'opération aient une importance.

3.Caractères et Chaînes de Caractères :

Pour manipuler des caractères individuels, il est nécessaire d'utiliser d'autres types de données, à savoir les types de caractères et les chaînes de caractères. En Pascal, les caractères et les chaînes de caractères font la distinction entre majuscules et minuscules. Le type de caractère, représenté par le mot-clé "char" en Pascal (également désigné par les termes "char," "single," ou "integer" pour indiquer les types), permet de stocker un seul caractère, qu'il s'agisse d'une lettre, d'un chiffre, ou de caractères moins courants tels que # ! ? , ; . : @, etc. Les chaînes de caractères et les caractères sont tous deux encadrés par des guillemets simples (' ').

4.Booléens :

Le type "Boolean" offre la possibilité de manipuler des données de type Booléen. Une telle donnée ne peut prendre que deux valeurs : FAUX (valeur 0) ou VRAI (valeur 1). Dans le langage Pascal, le mot-clé "not" permet de nier une valeur Booléenne. Ainsi, "NON VRAI"

équivaut à "FAUX", et "non FAUX" équivaut à "VRAI". Vous pouvez également utiliser les opérateurs Booléens classiques, qui sont des mots en Pascal, tels que "or" (ou logique inclusif), "XOR" (ou logique exclusif), et "and" (et logique), avec des données de type "Boolean". Ces opérateurs suivent les règles de priorité logique suivantes :

1. La négation **not** a la priorité la plus élevée.
2. Ensuite vient l'operateur **and.**
3. Enfin **or** et **xor** ont la même priorité .

En plus de ces opérateurs, il existe d'autres opérateurs liés aux Booléens, appelés opérateurs de comparaison (par exemple, + ou / sont des opérateurs simples). Comme leur nom l'indique, ils servent à comparer deux expressions, souvent des nombres ou des valeurs Booléennes. Ils sont utilisés de la manière suivante :

« Expression 1 » opérateur « Expression 2 »

Par expression , on entend n'importe quoi qui soit une donnée du programme ou quelque chose de calculable .

Le résultat de ces opérateurs est un booléen, et vaut donc **true** ou **false**.

5.Types Énumérés :

Les types énumérés sont largement employés dans Pascal Objet. Ils s'avèrent utiles lorsque des données doivent prendre un nombre limité de valeurs, chacune ayant une signification spécifique. Ces valeurs sont attribuées sous forme de noms, qui deviennent ensuite des mots clés dans le langage Pascal.

IV.2.3. Structure d'un Programme Pascal :

Cette section vise à expliquer la structure et l'organisation générale d'un programme rédigé en langage Pascal, sans entrer dans les détails spécifiques.

IV.2.3.1. Structure d'un Fichier Projet :

Le concept de fichier projet est principalement associé à Delphi, mais il est important de comprendre sa structure. L'intégralité du texte d'un fichier projet est générée par Delphi. Voici un exemple de code pour un programme nommé "premieressai" :

```
uses
  Forms,
  Principale in 'principale.pas' {forme 1};
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Forme1);
  Application.Run;
end.
```

Ce qui est écrit entre accolades (texte en italique) constitue un commentaire, généralement ignoré par Delphi. Un fichier projet est un élément du langage Pascal.

Un fichier projet est constitué de plusieurs blocs de texte, chacun ayant sa propre syntaxe. Les blocs sont séparés par des points-virgules (;), à l'exception du dernier bloc qui se termine par un point (.). Un fichier projet commence par le mot-clé "program" suivi de son nom et d'un point-virgule pour séparer le bloc suivant.

Le deuxième bloc d'un fichier projet est spécial et sert de liaison entre les différentes unités et le fichier projet. Il commence par le mot-clé Pascal "uses" (en français : "utilise") suivi d'une liste d'éléments séparés par des virgules et se termine par un point-virgule. Chaque élément de cette liste contient au moins le nom d'une unité, qui est le nom du fichier sans l'extension ".pas". Cette unité peut faire partie ou non du projet.

Par exemple, l'unité "Principale" est mentionnée dans le fichier projet mais n'est pas intégrée au projet. Elle fait partie des nombreuses unités fournies par Delphi et peut être utilisée selon les besoins.

La section finale du fichier avant "end" représente la partie la plus cruciale du fichier projet. C'est celle qui orchestre l'exécution de l'application. Cette partie débute avec le mot-clé

réservé "begin" (en français : "début") et se conclut par le mot-clé réservé "end". À l'instar du fichier projet, cette section est également structurée en blocs, où chaque bloc est séparé des autres par un point-virgule (bien que le point-virgule soit omis avant le premier bloc et après le dernier). Chacun de ces petits blocs constitue une instruction du programme.

En résumé, la structure d'un fichier-projet peut être récapitulée comme suit.

bloc de déclaration de nom : **Program** *nom_du_programme ;*
bloc d'utilisation d'unités : **uses** *liste_d_unites ;*
blocd'instriction : **begin** *Instructions* **end .**

IV.2.3.2. Structure d'une unité :

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
    Button1: TButton;
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;
var
  Form1: TForm1
implementation
{$R *.dfm}
end.
```

IV.2.3.3. Les Identificateurs :

Les constantes, variables, unités et programmes sont ce que l'on nomme des identifiants. Un identifiant doit répondre aux contraintes imposées par le langage Pascal :

- Il s'agit d'une séquence de caractères composée de lettres majuscules ou minuscules, de chiffres et du souligné (_). En revanche, les espaces, les tirets, les accents et tout autre caractère sont interdits.
- Sa longueur maximale raisonnable est de 50 caractères (bien que ce chiffre maximal puisse varier avec les différentes versions de Delphi).
- Le premier caractère ne peut pas être un chiffre.
- Les majuscules et minuscules sont considérées comme équivalentes. Cependant, il est préférable d'utiliser des minuscules autant que possible, en réservant les majuscules uniquement pour la première lettre des mots

1. Constantes et Variables :

Une constante est un nom associé de manière permanente à un identifiant. Ce dernier ne peut pas être modifié pendant l'exécution de l'application, devenant ainsi un mot clé Pascal reconnu par Delphi. Une variable est un nom attribué à une donnée de type spécifique, par exemple 'byte', 'single', 'char', 'string[10]', etc. Contrairement aux constantes, la valeur d'une variable peut évoluer pendant l'exécution de l'application.

2. Procédures et Fonctions :

Dans le langage Pascal, les instructions sont regroupées en blocs nommés procédures et fonctions, similaires aux blocs "uses" ou "vars". Cela permet de décomposer des tâches complexes en tâches élémentaires. L'ensemble de l'application constitue la tâche complexe, tandis que les procédures et les fonctions représentent les tâches élémentaires, chacune accomplissant une action spécifique et ciblée (comme répondre à un clic sur un bouton).

- **Procédure :**

Les procédures sont définies par un ou deux blocs de texte Pascal. Le premier bloc, facultatif, sert à déclarer la procédure. Le second bloc, obligatoire, constitue le corps de la procédure (son implémentation). Le bloc commence par le mot-clé réservé "procedure" qui indique clairement qu'il s'agit d'une procédure, suivi d'un identifiant qui est le nom de la procédure. Ce nom sera utilisé pour exécuter la procédure. Les paramètres, éventuellement entre parenthèses, sont des données utilisées dans la procédure.

procédure identificateur [(variable : type de variables)] :

les paramètres entre parenthèses sont des données intervenant dans la procédure.

- **Fonction :**

Les fonctions sont similaires aux procédures, mais elles ont la particularité de renvoyer un résultat final. La syntaxe diffère légèrement, en particulier pour la première ligne, qui devient :

Function identificateur [(paramètres)] :type_resultat ;

'Type_Résultat' indique le type de la valeur renvoyée par la fonction, semblable à une variable appelée "résultat." Cette valeur résultat est utilisable comme une variable, bien qu'elle ne soit pas explicitement déclarée. Après l'exécution de la dernière instruction de la fonction, la valeur de "résultat" devient le résultat de la fonction.

- **Appels de procédures et de fonctions :**

Quand vous souhaitez utiliser une procédure ou une fonction, la syntaxe à suivre est la suivante : Nom de la procédure ou de la fonction (Paramètres d'entrée).

Le "Nom de la procédure ou de la fonction" indique quelle action ou quel calcul vous souhaitez effectuer. Si ces procédures ou fonctions nécessitent des paramètres, vous devez leur fournir des valeurs appropriées dans l'ordre où ces paramètres ont été définis. Ces valeurs sont placées entre parenthèses et séparées par des virgules. Ces valeurs peuvent être des nombres, des chaînes de caractères, des constantes, des variables, des paramètres ou des résultats de calculs, tant que leur type est compatible.

En ce qui concerne les procédures, leur appel et leur exécution constituent une instruction à part entière. Cependant, pour les fonctions, leur appel peut être intégré dans une instruction

plus large. Le bloc de code qui contient cet appel se comportera comme une constante, dont le type sera celui du résultat retourné par la fonction.

IV.2.3.4.Type de données avancés de pascal objet :

Ce chapitre plutôt théorique ,est concacré à la suite de l'étude du language de pascal objet , Delphi ne sera utilisé icique pour vous permettre de manipuler les notion nouvelles .

Les type de données que nous connaissez actuellement , a savoir nombres entiers , à virgule , caractéres et chaines de caractéres , énumérés et booléen , suffisent pour de petit programmes mais deviendront insuffisants dans de nombreuses situation , Dautres types de donées plus élaborés devront alors etre utilisés

Ⅳ.2.3.5.Création de nouveaux types :

À partir de ce point , il va vous faloir connaître un nouveau bloc Pascal : le bloc de déclaration de type . Se bloc permet de définir les nouveaux types , comme les blocs de constantes et des variables .

Un bloc de déclaration de type peut se situer aux même endroit que les blocs **const** et **var** . Le bloc commence par contre par le mot Pascal réservé **type** et prossède la syntaxe suivant :

Type

Déclaration_de_type ;

{ . . .}

Déclaration_de_type ;

Chaque déclaration de type est de la forme :

Identificateur = Specification_du_type

Ou l'identificateur est un déficitateur non encore utilisé qui désignera le nouveau type. Spécifications des types peut prendre des vers se forme et au fur et à mesure des paragraphes ci-dessous , nous enverrons un certain nombre. Une habitude veut que tout l'identificateur de type personnalisé commençent par la lettre T majuscule suivi d'un nom descriptif Il est possible de donner pour spécification du type un type standard , tel 'integer' ou 'byte' ou

n'importe quel type connu . Les types ainsi définis pouvant être utilisé ensuite dans les blocs **const** et **var** de la manière suivante

```
Type

TChaineCourt = String [10] ;

Const

CBonj : TChaineCour =’Bonjour’ ;

var

InitialeNom : TChaineCourt ;
```

Ⅳ.2.3.6.Type ordinaux :

Les types ordinaux sont non pas un nouveau type mais un particularité pour un type. Certains type permettant des valeurs classées suivant un ordre. Ce type sont alors dits ordinaux. C'est le cas, parmi les types qui vous connaissez, de tout de tout les types entiers ('enteger','byte',...) Des booléen, des caractères (mais pas des chaînes de caractère) et des énuméres . Elle sera dorénavant mentionné lors de présentation de nouveau type s'ils sont ordinaux ou pas L'équipe ordinaux ont tout un sort de correspondance avec les nombres entiers. Chaque donnée (constante ou variable) d'un type ordinal possède ceux qui ont appelé une valeur ordinal, qui est donné par la fonction 'Ord' . Cette fonction fait partie d'un ensemble de fonction assez particulière qui acceptant des paramètres divers. En l'occurrence 'Ord' accepte n'importe quel constante ou variable de type ordinal. La fonction 'Ord' sera utilisé dans les paragraphe suivants, de même que nous reviendrons sur le type ordinaux

III.2.3.3 .Type intervalle :

Le type intervalle permet de définir un nouveau type ordinal personnalisé autorisant un intervalle de valeur ordinales, en donnant les valeurs extremales (son minimum et son maximum) . Ceci permettra de nombreuses choses notamment pour un autre type que nous allons bientôt voir.

Ce type s'écrit comme suit :

ValeurMinimale.. valeurMaximale

Où valeurMinimale et valeurMaximale sont du mêmes type ordinal. Il est a noter que la valeur ordinale de valeurMinimale , ou sinon le compilateur signalera une erreur. La déclaration d'un type ou d'un variable de ce type se font de la manière suivante :

Type

Nom_de_type=ValeurMinimale ..ValeurMaximale ;

Var

Nom_de_variable : ValeurMinimal.. ValeurMaximele ;

Ⅳ.2.3.7.type ensemble :

Le type ensemble permet de créer des ensembles à cardinal fini (le cardinal d'un ensemble et le nombre d'éléments dans cet ensemble) . Les ensembles en Pascal ne pouvent contenir que des constantes de type ordinal, dont la valeur ordinale et comprise entre 0 et 255 . Toutes les constantes entières entre 0 et 255 et conviennent donc , est-ce que les caractères et la plupart des types énumérés. On ne peut cependant pas mélanger les types à l'intérieur d'un ensemble. Le type des éléments étant décidé dans la déclaration du type. Le type ensemble s'écrit : **Set of** type_ordinal ;

Ou type_ordinal Designé au choix un nom de type ordinal ou spécifier directement un type ordinal. Voici des exemples

Type

TResultats= **set of** Byte ;

TNotes = **set of 0 ..**2O ;

TSupports = **set of** TTypeSupport ;

Dans l'exemple ci-dessus 'TResultat' pourra contenir des entiers de type 'byte','TNotes'pourra contenir des entiers entre 0 et 20, et 'TSupports' pourra contenir des éléments de type'TTypeSupport' il est alors possible de déclarer des variables de ces types

Ⅳ.2.3.8.Tableaux :

ce sont des type structurés qui permettent de stockes des données de meme type dans le structure indexée .ils se declarent avec mot-clé **array** et ce manipulent par accés aux éléments

Ⅳ.2.3.9.Enregistrement :

Ce sont des types structurées qui permettent de regrouper des donneés de types différent sous un meme nom . ils se declarent avecle mot-clé **record** et se manipulent par affectation direct

Ⅳ.2.4 Structures de programmation en pascal :

La syntaxe de l'instrection conditionnelle est la suivante :

if expression booléenne **then** instruction1 **else** instruction2

if, then et else sont des mots du langage Pascal. Cette instruction sert à effectuer un traitement uniquement lorsqu'une condition bien déterminée est satisfaite. Ainsi, si l'expression booléenne est évaluée à vrai, c'est instruction1 (une instruction au choix) qui est effectuée, et dans le cas contraire, c'est à dire si l'expression booléenne est évaluée à false, c'est instruction2 qui est effectuée. Notons que "else instruction2" est facultatif (il arrive qu'il n'y ait aucune action à réaliser).

Exemple, x, y et z étant des variables entières ou réelles quelconques :

```
if (x > y) and (x > z) then writeln('x est le plus grand')
else writeln('x n''est pas le plus grand') { différent de x est le + petit ! }
```

On peut emboîter les instructions conditionnelles. En cas d'ambiguïté, le else se rapporte toujours au if le plus proche.

Exemple :

```
if (10 < 5) then if (5 > 6) then writeln('premier cas') else writeln('deuxième cas')
```

Rien n'est écrit à l'écran car 10<5 est faux et le else se rapporte au deuxième if ! Présentons ces instructions plus lisiblement :

```
if (10 < 5) then if (5 > 6) then writeln('premier cas')

else writeln('deuxième cas')
```

Ⅳ.2.4.2.Structures itérative :

Les structures itératives de programmation en Pascal sont des instructions qui permettent de répéter un bloc de code un certain nombre de fois ou tant qu'une condition est vérifiée. Elles sont utiles pour parcourir des tableaux, effectuer des calculs récurrents, ou créer des boucles. Il existe trois types principaux de structures itératives en Pascal :

La boucle for : Elle permet de répéter un bloc de code pour une valeur du compteur allant d'un début à une fin, avec un pas fixe. La syntaxe est la suivante :

```
for compteur := debut to fin

do

begin

 // instructions à répéter

end;
```

Par exemple, pour afficher les nombres de 1 à 10, on peut écrire :

```
for i := 1 to 10 do

begin

 writeln(i);

 end;
```

La boucle while :

Elle permet de répéter un bloc de code tant qu'une condition est vraie. La condition est évaluée avant chaque itération, donc si elle est fausse dès le départ, le bloc de code n'est pas exécuté. La syntaxe est la suivante :

```
while condition do

 begin // instructions à répéter
```

```
end;
```

Par exemple, pour calculer la factorielle d'un nombre n, on peut écrire :

```
n := 5; // nombre dont on veut la factorielle

f := 1; // variable qui contiendra le résultat

while n > 0 do

begin f := f * n; // on multiplie f par n

n := n - 1; // on décrémente n

end;

writeln(f); // on affiche le résultat
```

La boucle repeat until :

Elle permet de répéter un bloc de code jusqu'à ce qu'une condition soit vraie. La condition est évaluée après chaque itération, donc le bloc de code est exécuté au moins une fois. La syntaxe est la suivante :

```
Repeat

// instructions à répéter

until condition;
```

Par exemple, pour demander à l'utilisateur son âge jusqu'à ce qu'il soit majeur, on peut écrire :

```
age := 0; // variable qui contiendra l'âge saisi par l'utilisateur

repeat

writeln('Quel est votre âge ?');

readln(age);

until age >= 18;

writeln('Vous êtes majeur.');
```

Ⅳ.2.4.3.Controle avancé des boucles :

Les instructions de contrôle de boucle modifient l'exécution de sa séquence normale. Lorsque l'exécution quitte une étendue, tous les objets automatiques qui ont été créés dans cette étendue sont détruits.

Pascal prend en charge les instructions de contrôle suivantes..

Déclaration de contrôle et description

1_déclaration break Met fin au loop ou case et transfère l'exécution à l'instruction immédiatement après l'instruction de boucle ou de cas.

2_continue déclaration Force la boucle à sauter le reste de son corps et à retester immédiatement son état avant de réitérer.

3_Exit : Elle permet de sortir d'une procédure ou d'une fonction, sans exécuter les instructions qui suivent. Elle est utile pour retourner une valeur ou terminer une procédure avant la fin du bloc

Ⅳ.3.Programation Orientéé Objets :

La programmation orientée objets (POO) est un paradigme de programmation qui permet de structurer le code en utilisant des concepts tels que les classes, les objets, l'héritage, le polymorphisme, les interfaces, etc. La POO facilite la réutilisation, la modularité et la maintenance du code , à l'aide du langage Pascal est possible grâce à des environnements de développement intégré (EDI) tels que Delphi ou Lazarus, qui offrent des fonctionnalités pour créer et manipuler des classes, des objets, des interfaces, de l'héritage, du polymorphisme, etc.

Ⅳ.3.1.Avantage de la POO :

Delphi est un environnement de développement intégré (EDI) qui permet de créer des applications Windows en utilisant le langage Pascal. Delphi supporte la programmation orientée objets (POO) depuis sa version 5, et offre plusieurs avantages pour la POO, tels que :

1_La simplicité : et la clarté du langage Pascal, qui facilite la compréhension et la maintenance du code.

2_**La richesse** : et la diversité des composants graphiques et non graphiques, qui permettent de réaliser des interfaces utilisateur attrayantes et interactives, ainsi que des fonctionnalités avancées comme l'accès aux bases de données, les communications réseau, le multithreading, etc.

3_ La compatibilité et la portabilité du code Pascal, qui peut être compilé avec d'autres compilateurs comme Free Pascal ou Lazarus, et qui peut être exécuté sur d'autres systèmes d'exploitation comme Linux ou MacOS.

4_ La rapidité et la performance : du code compilé, qui est directement exécutable sans avoir besoin de DLL supplémentaires ou d'interpréteurs.

Delphi est donc un outil puissant et efficace pour la POO, qui permet de développer des applications professionnelles et robustes sous Windows. Si vous souhaitez en savoir plus sur les avantages de Delphi pour la POO

Ⅳ.3.2.Définition des notions d'objet et de classe :

Objet : Il est impossible de parler de Programmation Orientée Objet sans parler d'objet, bien entendu. Tâchons donc de donner une définition aussi complète que possible d'un objet. Un objet est avant tout une structure de données. Autrement, il s'agit d'une entité chargée de gérer des données, de les classer, et de les stocker sous une certaine forme. En cela, rien ne distingue un objet d'une quelconque autre structure de données. La principale différence vient du fait que l'objet regroupe les données et les moyens de traitement de ces données.

1.2. Objet et classe Avec la notion d'objet, il convient d'amener la notion de classe. Cette notion de classe n'est apparue dans le langage Pascal qu'avec l'avènement du langage Delphi et de sa nouvelle approche de la Programmation Orientée Objet. Elle est totalement absente du Pascal standard. Ce que l'on a pu nommer jusqu'à présent objet est, pour Delphi, une classe d'objet. Il s'agit donc du type à proprement parler. L'objet en lui-même est une instance de classe, plus simplement un exemplaire d'une classe, sa représentation en mémoire. Par conséquent, on déclare comme type une classe, et on déclare des variables de ce type appelées des objets. Si cette distinction est à bien prendre en considération lors de la programmation en Delphi, elle peut toutefois être totalement ignorée avec la plupart des autres compilateurs Pascal. En effet, ceux-ci ne s'appuient que sur les notions d'objet et d'instance d'objet. Nous adopterons par conséquent ici ce point vue, qui simplifie le

vocabulaire et la compréhension. On pourra remarquer que FreePascal pour sa part définit une classe comme un "pointeur vers un objet ou un enregistrement".

3.Utilisation des objets :

Les objets se distinguent des autres variables. Comme les pointeurs, ils requièrent des opérations spéciales et leur usage obéit à certaines règles

Construction et destruction :

Deux mécanismes diffèrent dans la destruction d'objets en C++ et en Pascal Objet. Ce sont :

Les destructeurs appelés à cause d'exceptions déclenchées dans les constructeurs.

Les méthodes virtuelles appelées dans les destructeurs Les classes de style Delphi combinent les méthodes de ces deux langages.

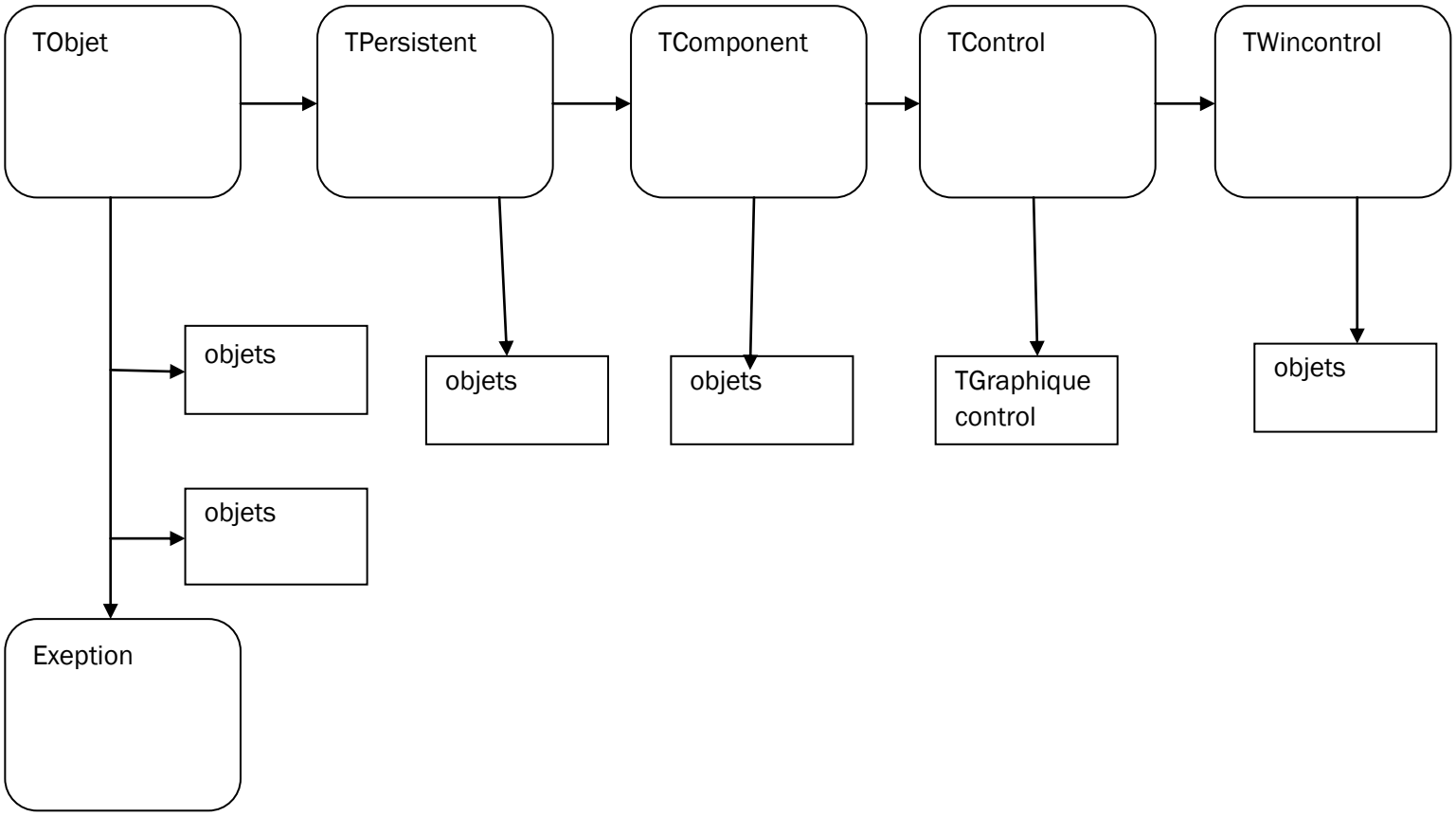
Construction et destruction :

Il existe des objets qui demandent deux actions spécifiques avant et après leur usage. Un objet se crée, s'emploie puis se supprime. La création s'effectue par une commande particulière que nous examinerons plus loin. La suppression se réalise aussi par une simple commande qui invoque le destructeur.

Ⅳ.3.3Notion avancées sur les classes :

Il ya deux aspect importants de la programmation orientée objet

Ⅳ.3.3.1.Hiérarchie des classes :



Figur. III 1 :Diagramme simplifie de la hiérarchie

Concepr général : La hiérarchisation est un concept qui permet de créer des classes à partir d'autres classes, en héritant de leurs attributs et de leurs méthodes. Une classe qui hérite d'une autre classe est appelée classe fille ou classe dérivée, et la classe dont elle hérite est appelée classe mère ou classe de base. La hiérarchisation permet de réutiliser du code, de simplifier la conception et de faciliter la maintenance des programmes

La notion de parenté entre les classes est au cœur du concept de hiérarchisation en programmation orientée objet. Chaque classe a un seul parent direct et unique. Une classe peut avoir autant de descendants qu'elle veut. Quand une classe hérite d'une autre classe, elle reçoit tout ce qui est déclaré par l'autre : c'est l'héritage. le principe de la hiérarchie des classes est en effet basé sur la relation parent-enfant ou plus exactement parent-descendant

La hiérarchisation des classes nécessite qu'il y ait un seul et unique ancêtre commun à toutes les classes : c'est la classe TObject. Cette classe est la racine de la hiérarchie des classes sous

Delphi, et toutes les autres classes en dérivent de façon plus ou moins directe (c'est-à-dire qu'elles sont des filles ou des petites-filles de TObject). La classe TObject définit les méthodes et les propriétés essentielles de tout objet, comme le constructeur, le destructeur, la gestion de la mémoire, le polymorphisme, la réflexion ou la gestion des exceptions

Ⅳ.3.3.2.Sections privées et publique d'une classe :

En fait, une classe peut comporter jusqu'à cinq sections. Ces cinq sections sont visibles au niveau du code source car elles sont séparées par des mots réservés à cet effet. Le début d'une section est généralement donné par un mot réservé, et sa fin est donnée par le début d'une autre section ou la fin d'une définition de classe (par le mot réservé **end**). La première section est délimitée au début par la classe (...) et terminée par le début d'une autre section ou la fin d'une déclaration de classe comme indiqué ci-dessous. Les deux autres sections commencent par les mots réservés **private** ou **public**.

Voici l'extrait d'un code montrant le début et la fin des trois sections :

Type

TfmPrince=class(TForm)

(section débutée apres class(…))

(section terminée par le début de la section ''private'')

Private

(section private débutée par le mot révervé''private'')

(private declarations)

{section private terminée par le début de la section ''public''}

Public

{section public débutée par le mot reservé ''public'' }

{public declarations }

{section public terminée par la fin de la classe (end) }

End ;

La première partie est réservée exclusivement à Delphi. Les sections suivantes commençant par le mot réservé private sont nommées « <section privée>> Classe : Tout ce qui y est écrit est inaccessible de l'extérieur. cette partie est idéale Placez des champs dont les valeurs ne doivent pas être modifiées de l'extérieur. La troisième et dernière section est appelée « section publique » : tout ce qui s'y trouve Les écritures sont accessibles en externe. Quant à la partie réservée à Delphi, tous les éléments qui y sont déclarés sont visibles comme s'ils étaient déclarés dans la partie publique.

Ⅳ.3.3.3.parametre de type objet :

Note générale : Ce sont les paramètres des procédures et des fonctions – dont le type est classe. Voici un exemple :

```
procedure TForml. ButtonlClick (Sender: TObject);

Begin

 end;
```

Cet extrait de code est le squelette de la procédure associée à l'événement OnClick. Il accepte un seul paramètre de type "TObject". Mais « TObject » est un objet, ce qui signifie que le paramètre nommé « Sender » est de type Object.

chapitre V "Application"

V.1.Conception :

La conception de notre application commence par le choix des composants fournis par Delphi dans notre interface (partie edetion) et la recherche d'un moyen de bien les utiliser.

V.1.1. Conception de la zone d'insertion :

V.1.1.1. Les composants utilisés :

1. Frame (cadre):

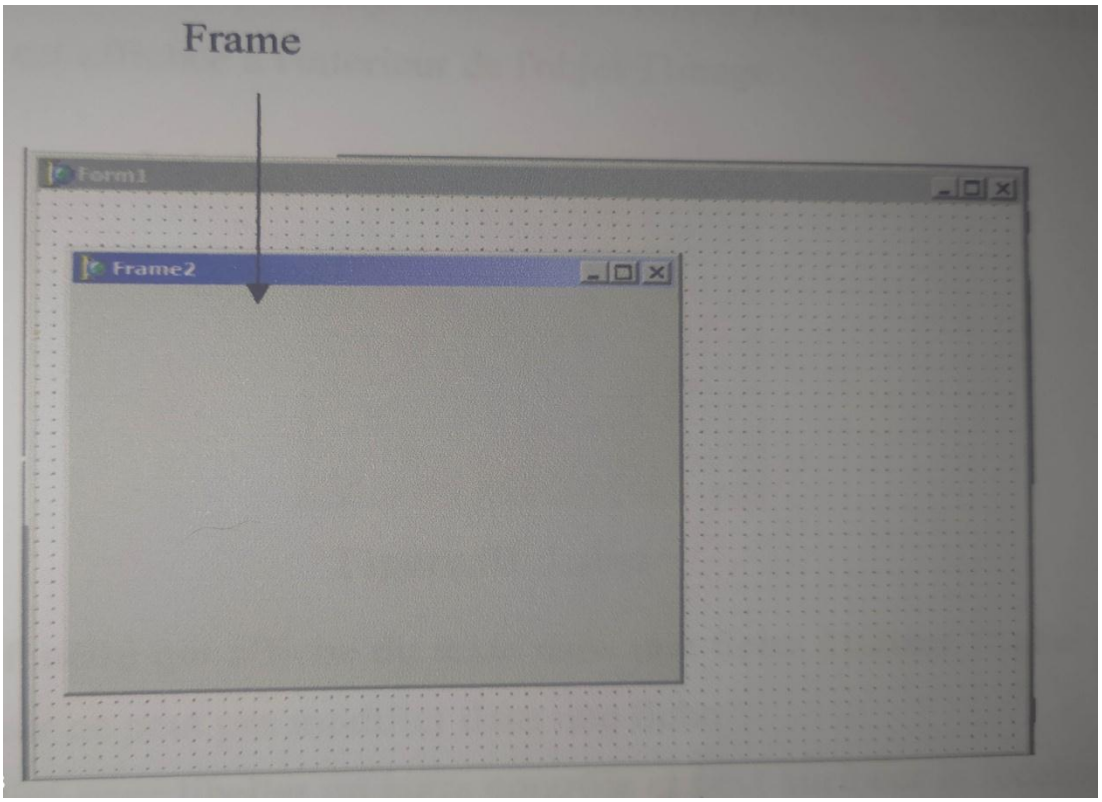


Figure. V 01 frame

Un TFrame (frame) est un conteneur de composants ; il peut être imbriqué dans un **fiche** ou un autre **cadre**. Les cadres, comme les formulaires, sont des conteneurs pour d'autres composants. Il utilise le même mécanisme de propriété qu'un formulaire pour instancier et détruire automatiquement les composants qu'il contient, et utilise la même relation **parent-enfant** pour synchroniser les **propriétés** des composants. Mais les cadres peuvent être imbriqués dans des **fiches** ou d'autres cadres, et ils peuvent être enregistrés dans la palette de composants pour une réutilisation facile. Une fois qu'un cadre est créé et enregistré, il

continue de fonctionner comme une unité et hérite des modifications apportées aux composants qu'il contient, y compris aux autres cadres. De plus, les images imbriquées continuent d'hériter des modifications apportées à leurs images dérivées.

2 .image dans le frame :

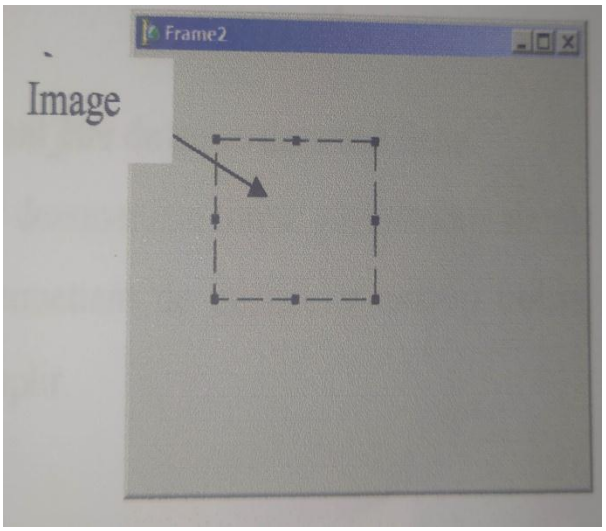
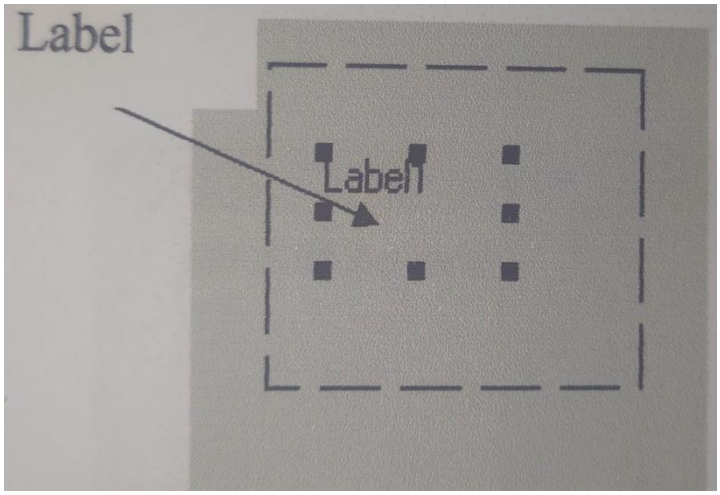


Figure Ⅴ ..02 image

TImage est un composant qui contient une image graphique placée sur un formulaire. Pour indiquer l'objet graphique (**bitmap**, **icône**, **métafichier**, etc.) que TImage doit afficher, on utilise l'objet TPicture de la propriété Picture. TPicture offre des propriétés et des méthodes pour manipuler l'image, comme la charger depuis un fichier, la supprimer de l'objet TImage ou la transférer à un autre contrôle. TImage dispose aussi de plusieurs propriétés qui permettent de contrôler la façon dont l'image est rendue dans l'objet TImage.



3.Label :

Figure. V .03 label

TLabel est un contrôle qui montre du texte sur une fiche sans avoir de fenêtre. Avec TLabel, on peut mettre du texte fixe sur une fiche. Ce texte peut servir à identifier un autre contrôle et lui donner le focus quand l'utilisateur appuie sur une touche de raccourci. Comme TLabel n'hérite pas de TWinControl, il n'a pas sa propre fenêtre et ne peut pas capter les entrées directes du clavier.

4.Les Shape :

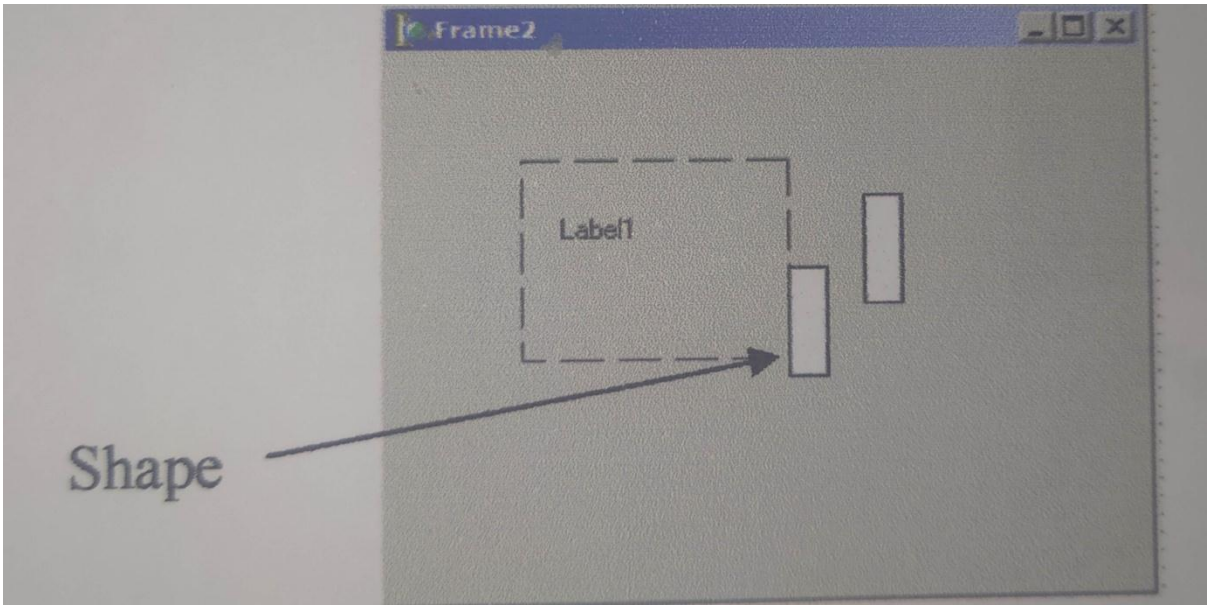


Figure V 04 shape

TShape est un contrôle qui permet de tracer une forme géométrique basique sur une fiche. En ajoutant un élément TShape à une fiche, on peut créer une forme géométrique sur cette fiche TShape dispose de propriétés qui définissent le style du contour et du fond de la forme.

V.1.1.1.2.L'ajustement des composants :

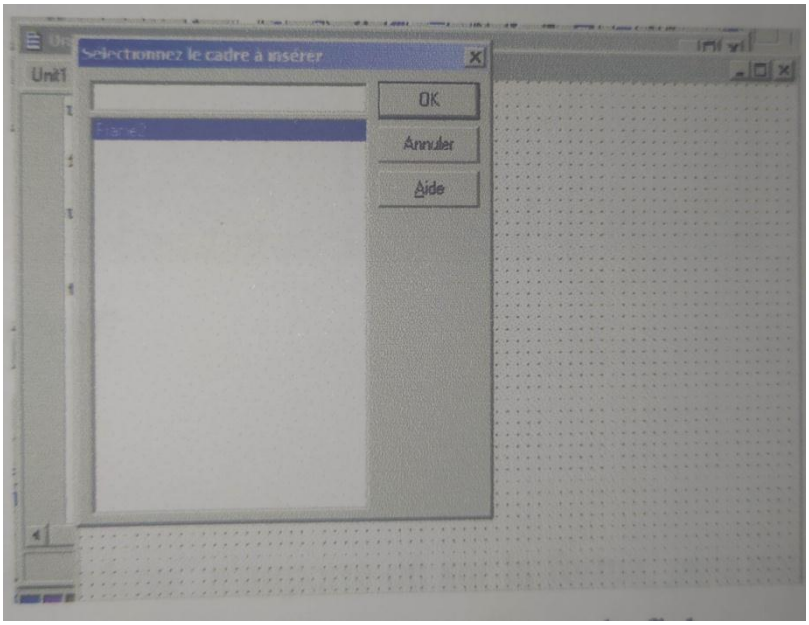


Figure V 05 shape L'ajustement des composants

V.1.2. Ajouter un menu :

Pour commencer, nous déposons un composant "MainMenu" sur la forme, à l'endroit que l'on veut, et nous cliquons deux fois dessus. Nous construisons les menus et sous menus courants :

1_Fichier (Nouveau, Ouvrir, Enregistrer, Enregistrersous, Quitter)

2_Instruction: (Insérer contacte, Insérer sortie)

3_Compilation

4_Option :

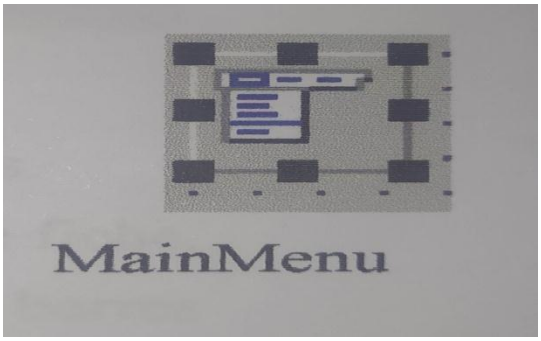


Figure V 05 mainmenu

1.Pour sélectionner le fichier à ouvrir :

Nous posons un composant dialogue "OpenDialog" sur la forme, où on veut. Nous cliquons deux fois dessus et nous écrivons le code Pascal objet dans la fenêtre d'édition.

2. Menu Enregistrer sous :

Nous mettons un composant "SaveDialog" sur la forme, à n'importe quel endroit; Nous cliquons deux fois dessus et nous rédigeons le code Pascal objet dans la fenêtre d'édition. et nous rédigeons le code Pascal objet.

3.Fichier Nouveau :

Il suffit d'utiliser la fonction Fermer sans se soucier de sa valeur de retour

3. **Menu Quitter** :

Il suffit de taper « close » pour quitter

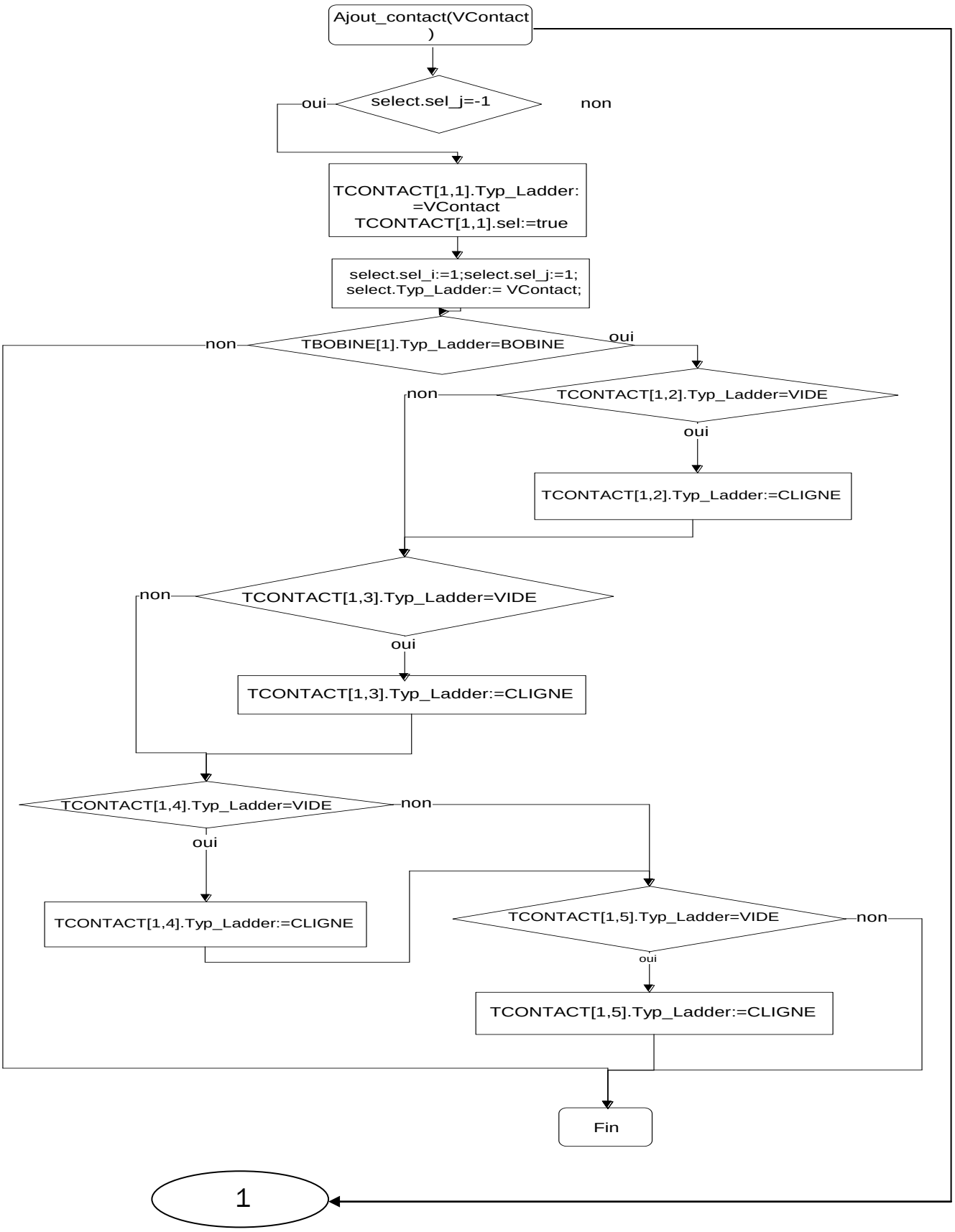
V.2.Partie intelligente :

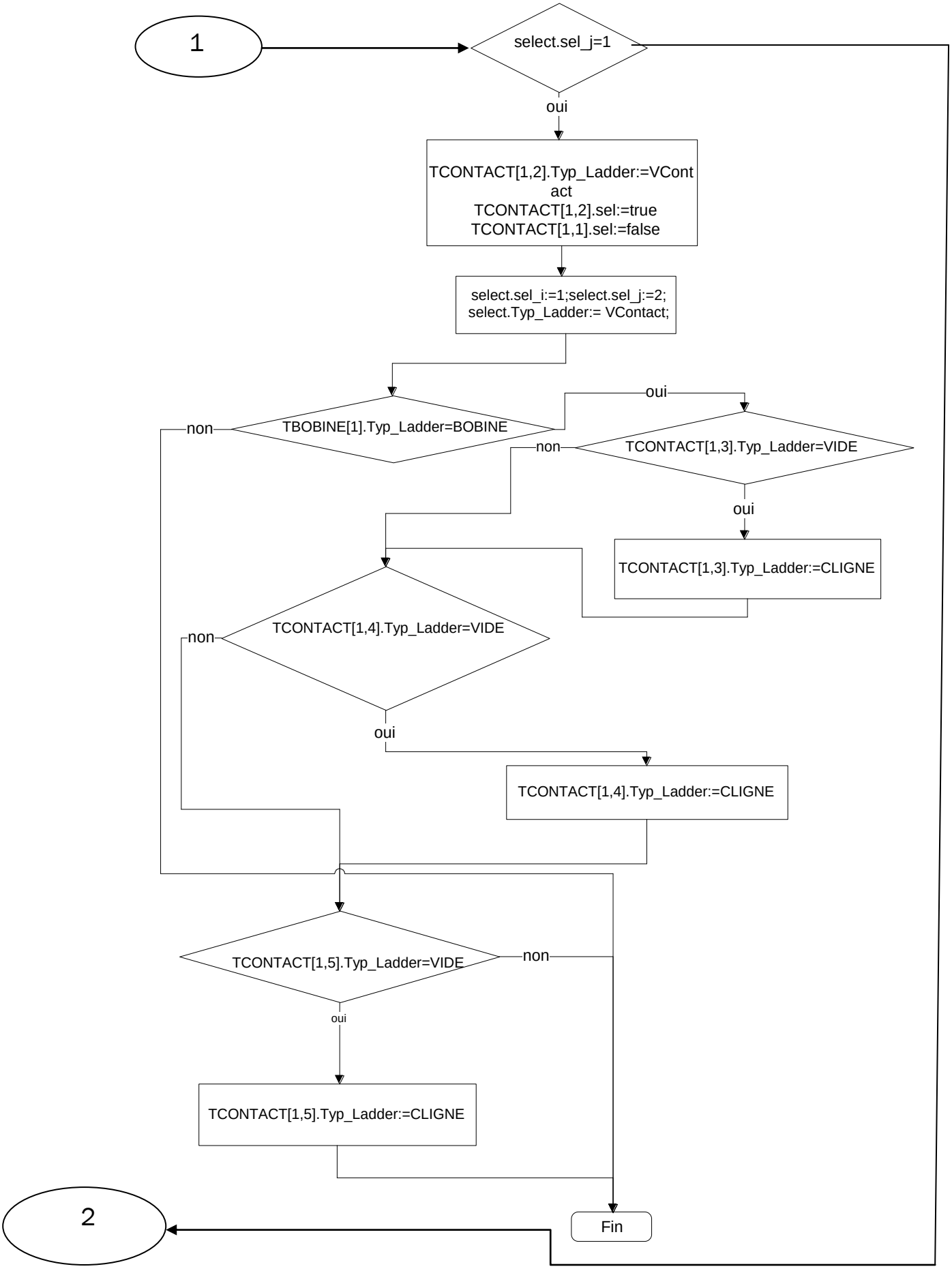
C'est la partie cachée de notre application, elle est composée en grande partie de programmes. C'est dans cette partie que le compilateur réalise les opérations logiques programmées. Il est recommandé que la meilleure façon de faire un programme avec le minimum d'erreurs est de se laisser guider par les organigrammes. Avant de commencer l'explication des organigrammes qui représentent les programmes et les sous programmes

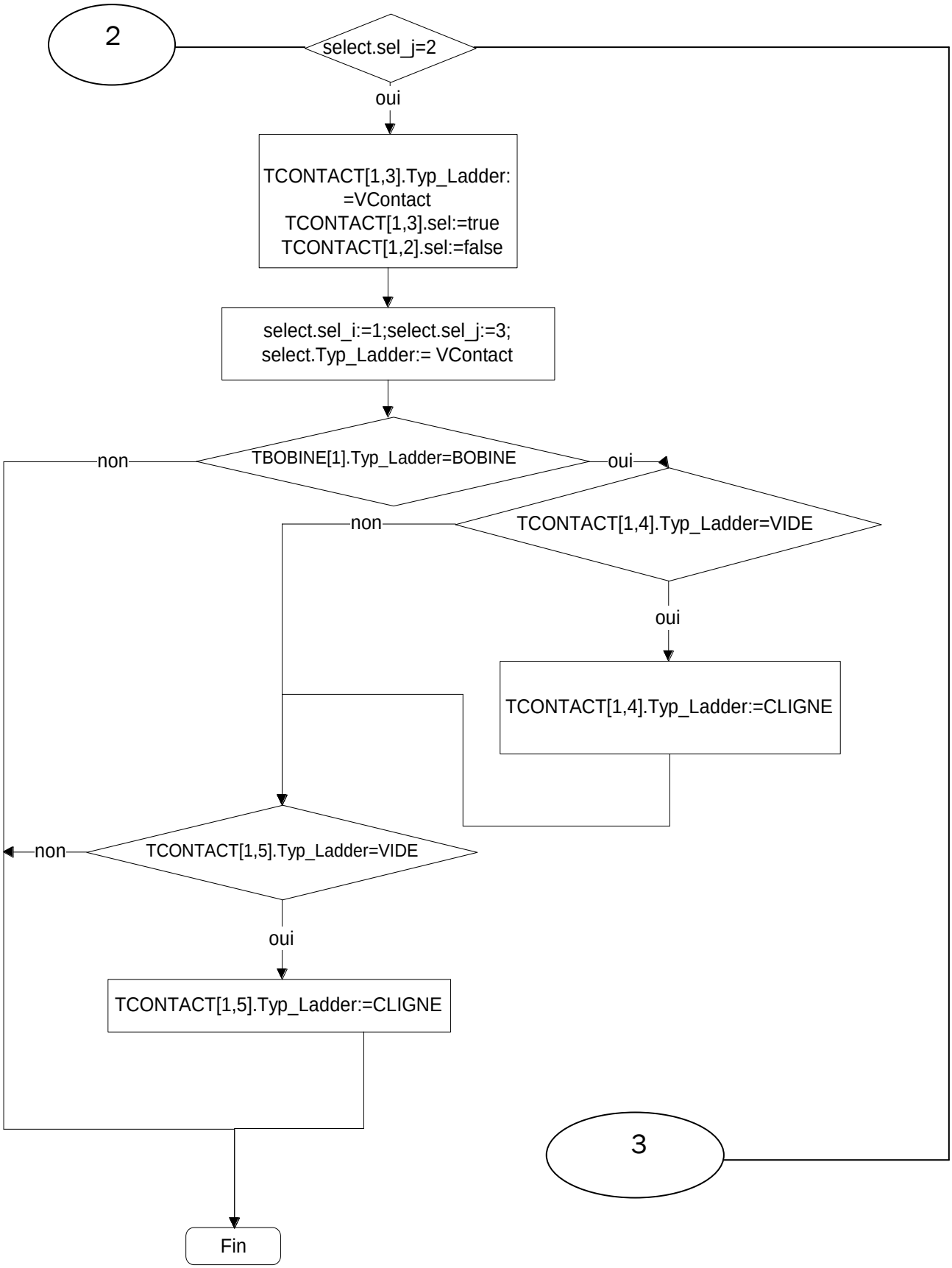
réalisés, on présente quelques symboles utilisés. Voici quelques organigrammes principaux réalisés: On différencie deux sortes d'organigrammes dans cette partie intelligente :

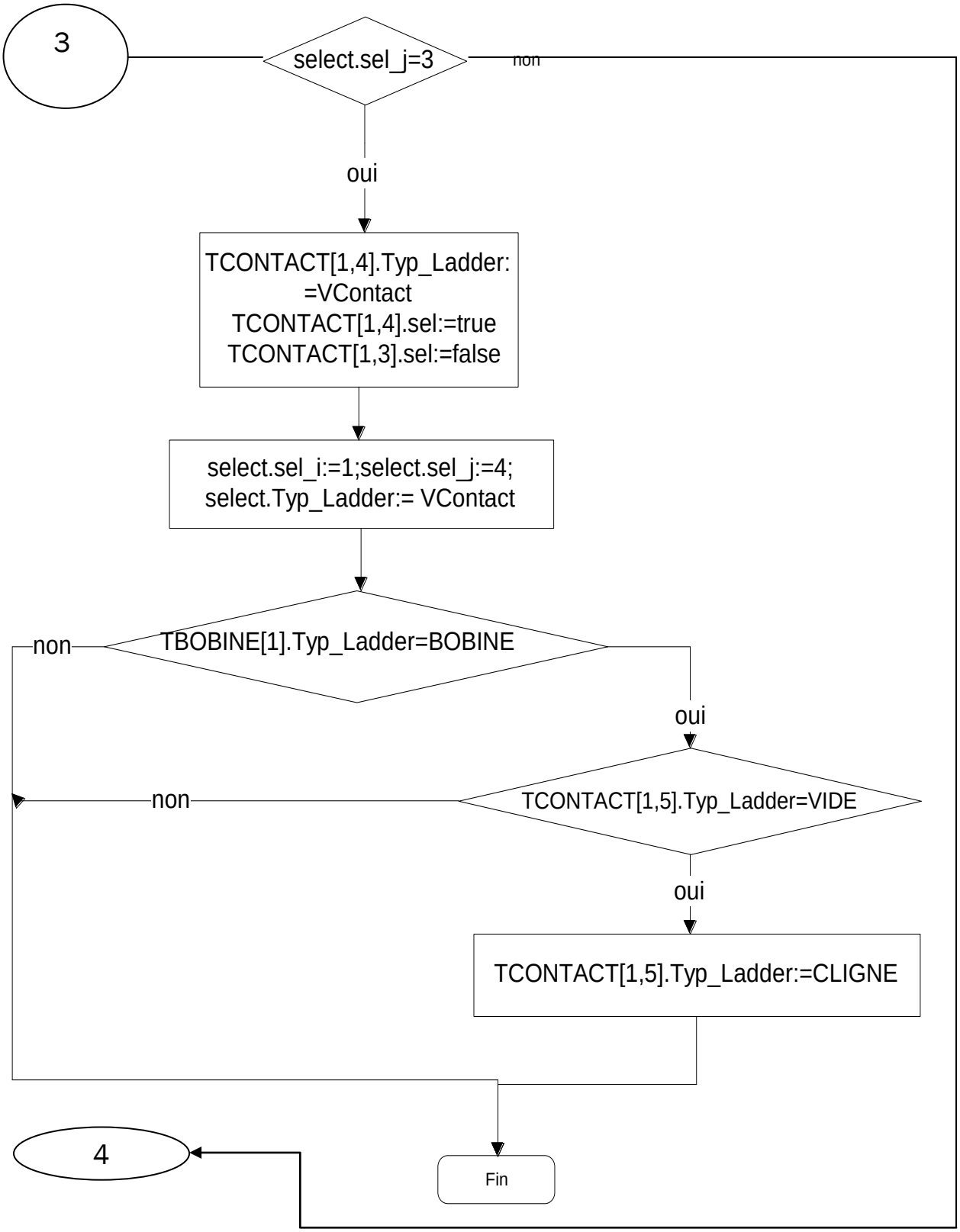
1.Organigramme <<ajoute_contact>> :

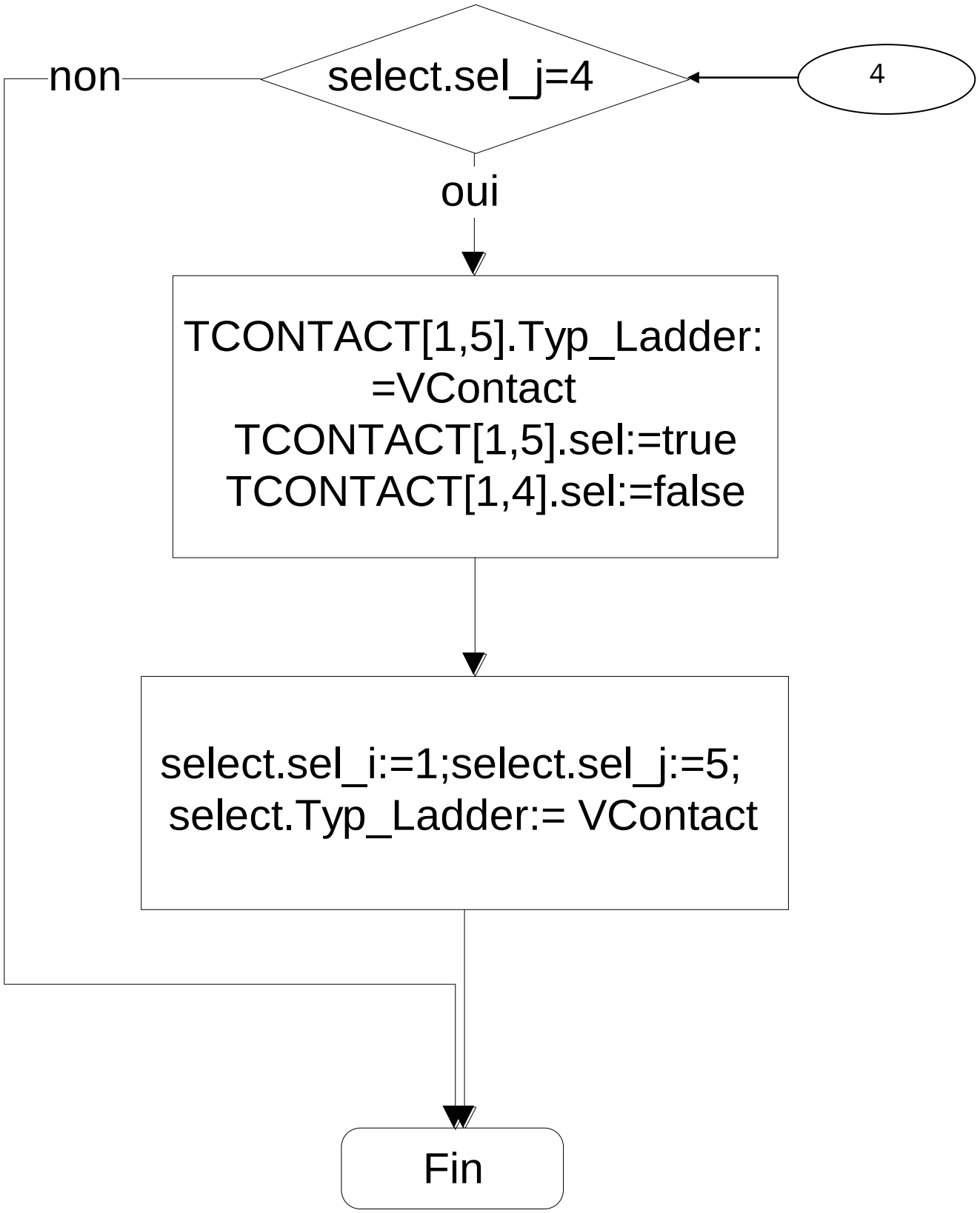
Cet organigramme offre la possibilité d'intégrer des contacts dans les cinq premiers cadres de chaque ligne, à condition que la sixième colonne contienne une bobine. Pour ajouter un contact, il est nécessaire de trouver un contact précédent dans la même ligne.


Ajout_contact(VContact)
select.sel_j=-1
oui
non
TCONTACT[1,1].Typ_Ladder: =VContact
TCONTACT[1,1].sel:=true
select.sel_i:=1;select.sel_j:=1;
select.Typ_Ladder:= VContact;
TBOBINE[1].Typ_Ladder=BOBINE
non
oui
TCONTACT[1,2].Typ_Ladder=VIDE
non
oui
TCONTACT[1,2].Typ_Ladder:=CLIGNE
TCONTACT[1,3].Typ_Ladder=VIDE
non
oui
TCONTACT[1,3].Typ_Ladder:=CLIGNE
TCONTACT[1,4].Typ_Ladder=VIDE
non
oui
TCONTACT[1,4].Typ_Ladder:=CLIGNE
TCONTACT[1,5].Typ_Ladder=VIDE
non
oui
TCONTACT[1,5].Typ_Ladder:=CLIGNE
Fin
1


1
select.sel_j=1
oui
TCONTACT[1,2].Typ_Ladder:=VContact
TCONTACT[1,2].sel:=true
TCONTACT[1,1].sel:=false
select.sel_i:=1;select.sel_j:=2;
select.Typ_Ladder:= VContact;
oui
non
TBOBINE[1].Typ_Ladder=BOBINE
non
TCONTACT[1,3].Typ_Ladder=VIDE
oui
TCONTACT[1,3].Typ_Ladder:=CLIGNE
non
TCONTACT[1,4].Typ_Ladder=VIDE
oui
TCONTACT[1,4].Typ_Ladder:=CLIGNE
TCONTACT[1,5].Typ_Ladder=VIDE
non
oui
TCONTACT[1,5].Typ_Ladder:=CLIGNE
Fin
2


2
select.sel_j=2
oui
TCONTACT[1,3].Typ_Ladder:
=VContact
TCONTACT[1,3].sel:=true
TCONTACT[1,2].sel:=false
select.sel_i:=1;select.sel_j:=3;
select.Typ_Ladder:= VContact
non
TBOBINE[1].Typ_Ladder=BOBINE
oui
non
TCONTACT[1,4].Typ_Ladder=VIDE
oui
TCONTACT[1,4].Typ_Ladder:=CLIGNE
non
TCONTACT[1,5].Typ_Ladder=VIDE
oui
TCONTACT[1,5].Typ_Ladder:=CLIGNE
3
Fin


3
select.sel_j=3
non
oui
TCONTACT[1,4].Typ_Ladder:
=VContact
TCONTACT[1,4].sel:=true
TCONTACT[1,3].sel:=false
select.sel_i:=1;select.sel_j:=4;
select.Typ_Ladder:= VContact
non
TBOBINE[1].Typ_Ladder=BOBINE
oui
non
TCONTACT[1,5].Typ_Ladder=VIDE
oui
TCONTACT[1,5].Typ_Ladder:=CLIGNE
4
Fin



Organigramme ajoute_contact

Les frame des autre ligne en a le meme code source et meme organigrame

2.code source <<ajoute_bobine>> :

Ce programme offre la possibilité d'ajouter une sortie, qu'il s'agisse d'une bobine normalement ouverte ou d'une bobine normalement fermée, dans le sixième cadre de chaque ligne.

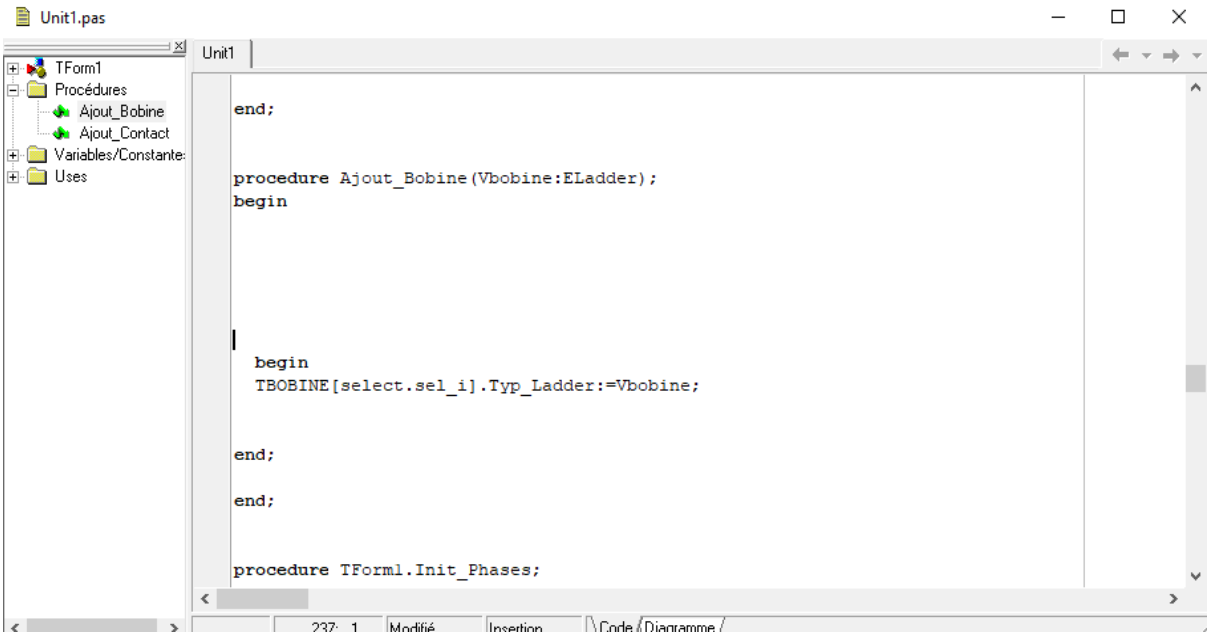
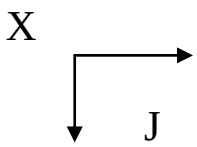


Figure V 06 code source ajoute bobine

4. Tableau des frames



X21	22	23	24	25	26
27	28	29	210	211	112
213	214	215	216	217	218
219	220	221	222	223	224
225	226	227	228	229	230
231	232	233	234	235	236
237	238	249	240	241	242
243	244	245	246	247	248

Ce TABLEU exmplique le programme qui offre la possibilité d'intégrer un ensemble de composants de type "frame" sous une disposition en grille de 8 lignes et 6 colonnes, numérotées de 1 à 48. La sixième colonne est réservée aux sorties, également appelées bobines, tandis que les cinq premières colonnes sont dédiées aux entrées, souvent désignées sous le terme "contacts".

V.2. Exécution : Apes avoir cliqué sur le bouton « exécuter » de Delphi

Figure.11: L'interface générale :

L'interface utilisateur de notre application se compose de :

I. Menu :

1. Fichier : Nouveau, ouvrir, enregistrer, configuration de l'impression quitter

2. Instruction: Insérer contact (normal, négative), Insérer sortie

3. Option: Couleur

4. Compilation :

II. Barre d'outil (figure. 12):

1.Nouveau

2.Ouvrir

3.Sauvgarder

III.Zone d'insertion :

C'est la partie délimitée par deux barres verticales (figure. 11).

IV. Une boite de dialogue: qui se compose : 1. NOM: pour attribuer une variable à un contact ou une sortie. 2. PIN entrer et sortie : pour définir le contacte en tant que entrée ou sortie

3.un checkbox: pour mettre la barre (/) à un contact si on veut sa négation.

4. Bouton : 1. OK: pour confirmer un choix

2. Annuler: pour annuler un choix.

Conclusion

La programmation est un univers où l'on dépeint les données, les résultats, et leurs relations. C'est une discipline scientifique et technique qui nécessite une formation sérieuse.

Pour créer un logiciel avec Delphi, on doit simultanément concevoir deux parties, l'une visible et l'autre invisible. Ces deux parties ne sont pas indépendantes, mais étroitement liées. La partie visible agit comme une façade, cachant le travail accompli en coulisses, suivant ainsi le modèle de l'iceberg. L'utilisateur ne doit percevoir que la partie émergée, ignorant ce qui se passe en dessous. Cette zone cachée est le domaine des programmeurs et des développeurs, comme nous le sommes maintenant. Bien qu'invisible, cette partie immergée joue un rôle prépondérant, contrôlant ce qui est visible à la surface tout en permettant à l'ensemble de flotter sans couler.

La notion d'objet peut sembler complexe, mais une fois ses principes compris, tout devient plus clair. Après avoir accompli ce travail, nous pouvons conclure que nous avons approfondi nos compétences en programmation orientée objet grâce à Delphi 7, en comprenant des concepts tels que les objets et les champs. Nous avons également exploré les caractéristiques, le langage et les performances de ce logiciel.

Cette expérience nous a permis de faire un saut significatif dans la programmation en Pascal. Nous avons acquis une méthodologie pour résoudre des problèmes avec une flexibilité accrue dans l'interprétation des algorithmes en code Pascal. Notre projet de fin d'études a ouvert les portes du monde du génie logiciel, des applications informatiques, et de la programmation en tant que discipline scientifique et économique qui est devenue un produit industriel essentiel.

En perspective, nous encourageons les futurs ingénieurs à poursuivre sur cette voie de la programmation, en vue de concevoir de nouvelles applications à des fins pédagogiques ou professionnelles.

Bibliographie

1. Y.LECOURTIER,B,SAINT-JEAN :

Introduction aux automatismes industriels ,MASSON,1989.

2.W. Bolton ; « Les automates programmable industriels », Edition Dunod ,2010

3. G. Michel , « Les API. Architecture et application des automates programmables industriels », Edition Dunod , 1998.

4. Support de cours : Systèmes automatisés . Bacem JRAD ,2011-2012

5.Daniel Bouteille , 'Les automates programmables' CEPADUES-Edition.

6.BELAID, Mohamed Chrif . Delphi ,de l'initiation à la maitrise, Alger , Editions Pages bleues internationales,2005.

7.Guide du développeur Delphi 7,PDF.

8.S. Graine, Delphi 7 , conception de bases de données , Tizi-Ouzou , Edition l'abeille, 2006.

9.Titre : le langage pascal appliqué à l'algorithmique

Auteur : CLAUDE Bauer et PIERRE Vincenti.

10.[Su97] Su (zhondong)-Automatic Analysis of relay Ladder logic programs.

Annexe

1. Exemples des systèmes automatisés :

Système	Matières d'œuvre		Valeur Ajoutée	Contexte	
	M.O.E	M.O.S			
Entreprise de presse	Informations .	Revues, jounaux...	Mise en forme, .commentaire, Diffusion	Public visé, lois sur la presse,	
Centrale électrique	Energie : pétrole, . charbon, gaz,...	Energie . électrique	Conversion sous une forme plus aisément distribuable	Prix des matières premières, consommation,...	
Usine d'assemblage (automobile, électronique)	Composants, . pièces détachées	Constituants et équipements	assemblage	Consommation, prix, concurrence, sous-traitance, ...	
Centre de soins (système de santé)	Malades	Individuels soignés	Soins, bonne . . . santé	Sécurité sociale, moyens et personnels de santé,	

2. Processus :

Un processus peut être considéré comme un système organisé d'activités qui utilise des ressources (personnels, équipements, matériels et machines, matières premières et informations) pour transformer des éléments entrants en éléments de sortie dont le résultat final attendu est un produit.

3. - Matières d'œuvre :

Une matière d'œuvre peut se présenter sous plusieurs formes. Par exemple :

- Un PRODUIT, c'est à-dire de la matière, à l'état solide, liquide ou gazeux, et sous une forme plus ou moins transformée :

- ✓ des objets techniques : roulement, moteur, véhicule, …etc
- ✓ des produits chimiques : pétrole, matière plastiques…etc)
- ✓ des produits textiles : fibre, tissu, vêtements…etc)
- ✓ des produits électroniques : transistor, puce, microprocesseur, automate) programmable,…etc

- De l'ENERGIE

- ✓ sous forme : électrique, thermique, hydraulique,…etc.
- ✓ qu'il faut : produire, stocker, transporter, convertir, utiliser,…

- De l'INFORMATION

- ✓ sous forme écrite, physique, audiovisuelle,…etc.
- ✓ qu'il faut : produire, stocker, transporter, transmettre, communiquer, décoder, utiliser,…etc.

4. Valeur ajoutée :

La valeur ajoutée à ces matières d'œuvre est l'objectif global pour lequel a été défini, conçu, réalisé puis éventuellement modifié, le système. Cette valeur ajoutée peut résulter par exemple :

- Une MODIFICATION PHYSIQUE des matières d'œuvre :

- ✓ traitement mécanique : usinage, broyage,..etc
- ✓ traitement chimique ou biologique ;
- ✓ conversion d'énergie ;
- ✓ traitement thermique ;

- D'une MISE EN POSITION particulière, ou d'un TRANSFERT, de ces matières d'œuvre :

- ✓ Manutention, transport, stockage ;

■ D’un prélèvement D’INFORMATION sur ces matières d’œuvre :

- ✓ contrôle, mesure, lecture,..etc

5- Contexte et valeur ajoutée

● La nature, la quantité et la qualité de la valeur ajoutée peuvent varier pour tenir compte de l’évolution des besoins de la société dans laquelle s’insère le système. Ce qui peut conduire à modifier le système, voire l’abandonner pour en construire un nouveau.

■ L’environnement, c’est à-dire le CONTEXTE physique, social, économique, politique,…joue un rôle essentiel dans le fonctionnement du système et influe sur la qualité et/ou la quantité de la Valeur ajoutée.

6. Autre menus :

Chacun des autres menus possède des commandes intéressantes, mais ce serait assez fastidieux de les lister toutes .

7. Menu fichier :

Nouveau…	permet de créer un nouvel élément. Cet élément peut être beaucoup de choses. Parmi lesquels un projet, une unité,(un .PAS sans .DFM correspondant), une fiche, (un .PAS et un. DFM associés), et beaucoup d'autres choses qui dépassent le cadre de ce guide. (Certains choix, comme ‘ composants’, ‘cadres’, .DLL, seront abordées plus loin).
Ouvrir	Permet d'ouvrir un fichier : On ouvre un projet en ouvrant son fichier .DPR unique. On ouvre une fiche en ouvrant le fichier .DFM Ou le fichier .PAS correspondant. On ouvre une unité (sans fiche) en ouvrant le .PAS qui la contient. Enfin, tout fichier texte peut être ouvre dans Delphi Selon des besoins divers et varié
Enregistrer	Permet d'enregistrer l'élément en cours d'édition. Cet élément est soit une unité (cf. Description de l'éditeur de code, soit une fiche. Si l'élément n'a encore pas été enregistré, un nom vous sera demandé. Il peut être temps de créer un répertoire pour le projet et de réfléchir aux noms du futur fichier. EXE du logiciel si on enregistre Le. DPR(car il est rappelé que ces deux fichiers ont le même nom de base).
Enregistrer Sous…	Comme enregistrer, mais permet en plus de définir un nouveau nom.
Tout Enregistrer	Permet d'enregistrer d'un coup tous les fichiers d'un projet ouvert. Des noms de fichiers vous seront demandés si c'est nécessaire.
Fermer	Ferme l'élément en cours d'édition. Propose l'enregistrement si c'est nécessaire.
Tout fermer	Ferme tous les éléments ouverts dans le Delphi. Dans le Delphi 2 à 4. Si j'ai bonne mémoire, c'est plutôt fermer projet.
Quitter	Quitte Delphi.

8. Le composant :

Un projet développé avec Delphi se compose de divers éléments connus sous le nom de "composants". Ces composants représentent des programmes autonomes (de petits projets) qui exécute des tâches simples ou des fonctions spécifiques. Ces programmes sont créés et ensuite ajoutés à une palette en vue d'une possible réutilisation ultérieure.

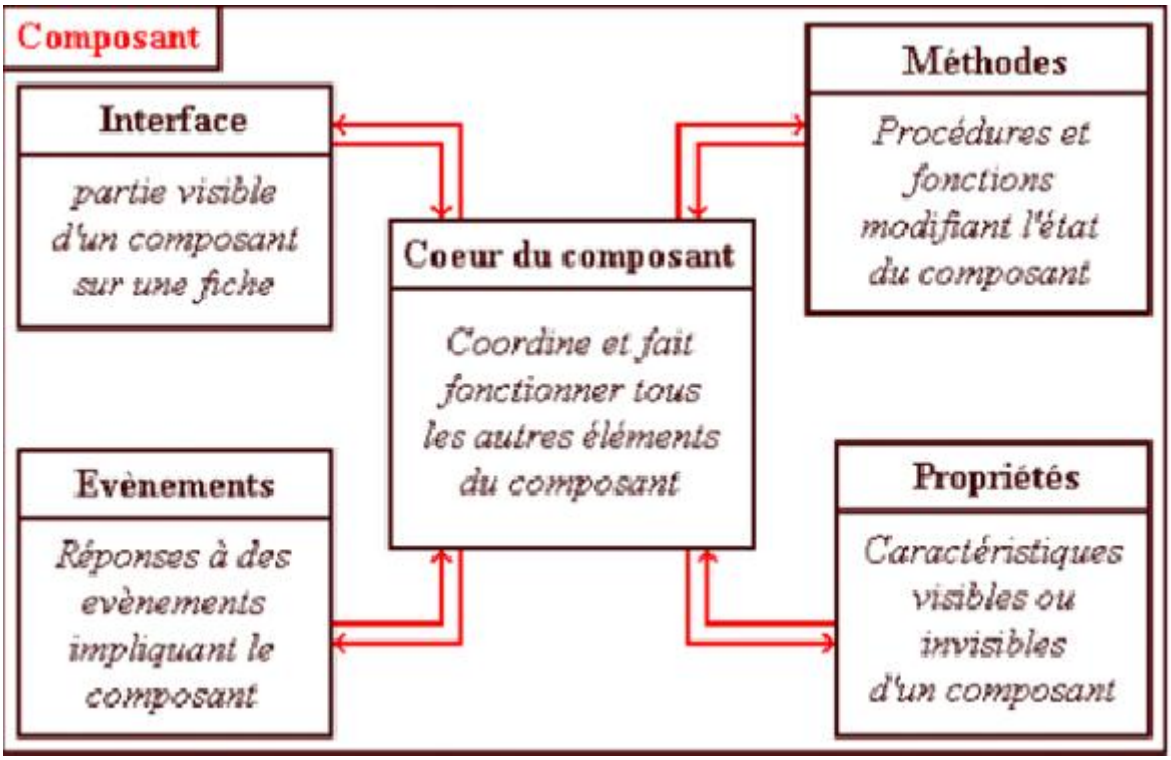


Figure : structure interne d'un composant

Tableaux à une seule dimension :

Un tableau à une seule dimension se déclare de la façon suivante :

array[intervalle] of type;

où **array** définit un tableau. les crochets entourent les données de dimension du tableau. Pour un tableau à une seule dimension, un seul paramètre est donné, de type intervalle (il est possible d'utiliser d'autres choses que les intervalles, Vient ensuite le mot réservé of qui est suivi

du type des éléments du tableau. Cet élément est un type. Tous les types vus jusqu'à présent sont acceptés, y compris d'autres tableaux. Il faudra cependant, pour certains types comme les ensembles, déclarer d'abord un type ensemble, puis utiliser ce nom de type pour déclarer le tableau. En principe, tous les types sont autorisés, du moment que la taille du tableau reste en dessous d'une limite dépendant de la version de Delphi.

Il est possible de déclarer des types, des variables et des constantes de type tableau (c.f. exemple ci-dessous). Pour accèder à un tableau, il suffit de donner le nom de la variable tableau de la constante tableau. Pour accèder à un élément, on donne le nom du tableau suivi de l'indice entre crochets.

Tableaux à plusieurs dimensions :

Vous aurez parfois besoin de tableaux à plusieurs dimensions (les tableaux à 2 dimensions sont similaires aux tableaux à double entrée que vous connaissez certainement déjà par exemple dans Microsoft Word, mais Pascal permet d'avoir plus de 2 dimensions). Je ne vous donnerai pas de preuve de leur utilité, vous en connaissez certainement déjà. Un tableau multidimensionnel se déclare ainsi :

array[intervalle1, intervalle2, ...] of type;

où intervalle1 est la première dimension du tableau, intervalle2 la deuxième, ... et type est encore le type des éléments stockés dans le tableau.

Les types, variables et constantes se déclarent en utilisant cette syntaxe. Pour donner des valeurs aux éléments d'une variable tableau, on utilise la syntaxe suivante :

nom_du_tableau[indice1, indice2, ...] := valeur;

où nom_du_tableau est le nom d'une variable de type

tableau, indice1 est l'indice dans la première dimension, indice2 dans la deuxième, ... et valeur la valeur (dont le type est donné dans la déclaration du tableau) à stocker dans cette case.

Pour fixer la valeur d'une constante de type tableau multidimensionnel, c'est un peu plus compliqué. Considérons un tableau de dimension 3 : on devra faire comme si c'était en fait un tableau à 1 dimension dont chaque case contient un tableau à 1 dimension, qui lui-même est de dimension 1. Voyez l'exemple ci-dessous qui récapitule tout ce que nous venons de dire (Attention, la première instruction, à savoir l'affectation d'un tableau à une variable tableau, est possible à partir de Delphi 4 seulement, si vous n'avez que Delphi 2 ou 3, il vous faudra attendre que nous ayons vu les boucles 'for' pour passer outre ce genre de difficulté).

Notions avancées sur les tableaux

Nous nous sommes limités jusqu'à présent pour les indices des tableaux à des intervalles. Ce n'est que l'une des possibilités offertes : la plus simple. Il est en fait possible d'utiliser tout type ordinal en tant qu'indice. Le cas le plus intéressant à examiner est celui des types énumérés : il sera possible de déclarer un tableau tel que celui-ci :

```
array[tsDisqueDur..tsDVDRom] of Char;
```

Mais il sera encore plus intéressant d'utiliser des noms de types énumérés en guise d'indices. Ainsi, la déclaration suivante permet d'utiliser un tableau dont chaque case est une valeur de type TTypeSupport :

```
type TTabSupport = array[TTypeSupport] of string;
```

Tableaux de taille dynamique :

Remarque : Les tableaux de taille dynamique sont utilisables à partir de Delphi 4.

Un tableau dynamique à une dimension est de la forme :

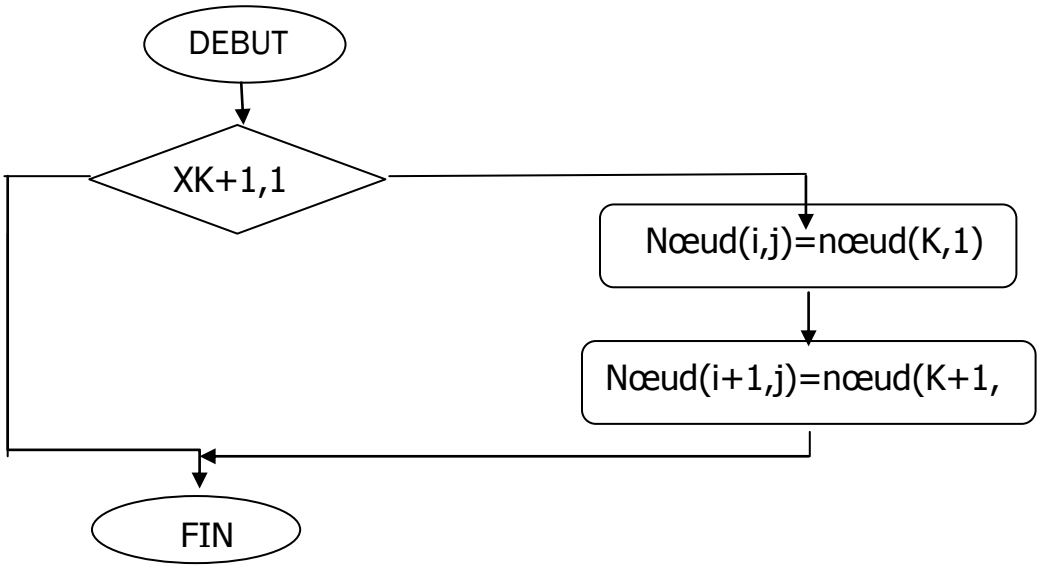
array of *type_de_base*

où *type_de_base*, comme pour les tableaux non dynamiques, est le type des éléments stockés dans le tableau. Vous remarquez que la partie qui donnait auparavant les dimensions, les bornes et les limites du tableau (la partie entre crochets) est justement absente pour les tableaux dynamiques.

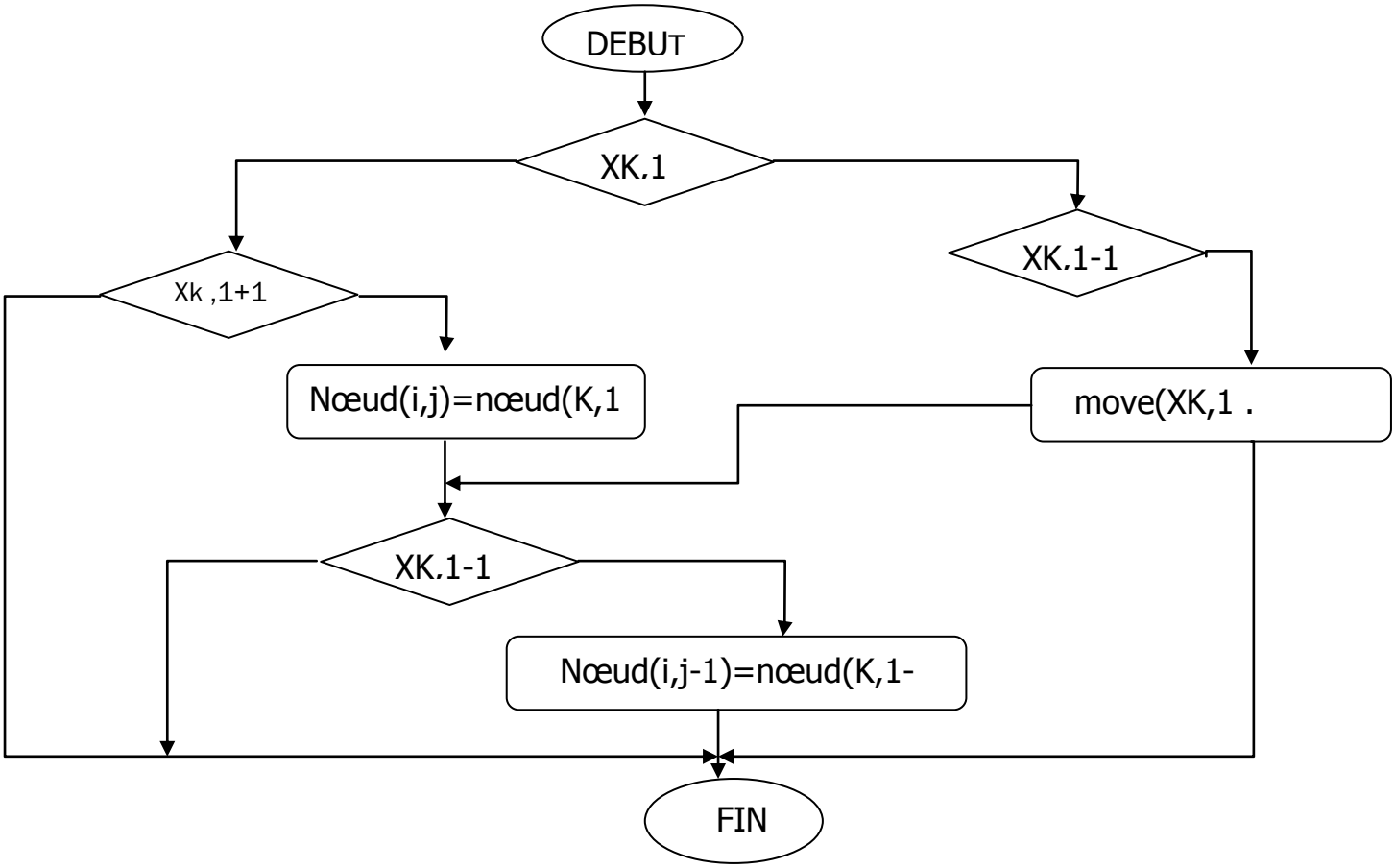
Ces derniers tableaux sont plus complexes à manipuler que les tableaux standards : déclarer une variable de type tableau dynamique ne suffit pas, comme dans la plupart des autres cas de variables, pour que la variable soit utilisable. Du fait que leur taille n'est pas fixe, il faudra la gèrer en même temps que le tableau lui-même et donc fixer le nombre d'éléments avant la première utilisation du tableau au moyen d'une procédure : 'SetLength'. 'SetLength' accepte plusieurs paramètres : le premier est une variable de type tableau dynamique, le deuxième paramètre est le nombre d'éléments que le tableau doit pouvoir contenir. Par la suite, lorsque vous désirerez redimensionner un tableau dynamique, il suffira de rappeler "SetLength" avec une taille différente. Il est également possible de connaître la taille actuelle d'un tableau dynamique en utilisant la fonction 'Length'. 'Length' accepte un unique paramètre qui doit être une variable de type tableau dynamique.

les algprithmes des programmes complementaires :

Voisinage de AND ou (Shift_AND_Del) :



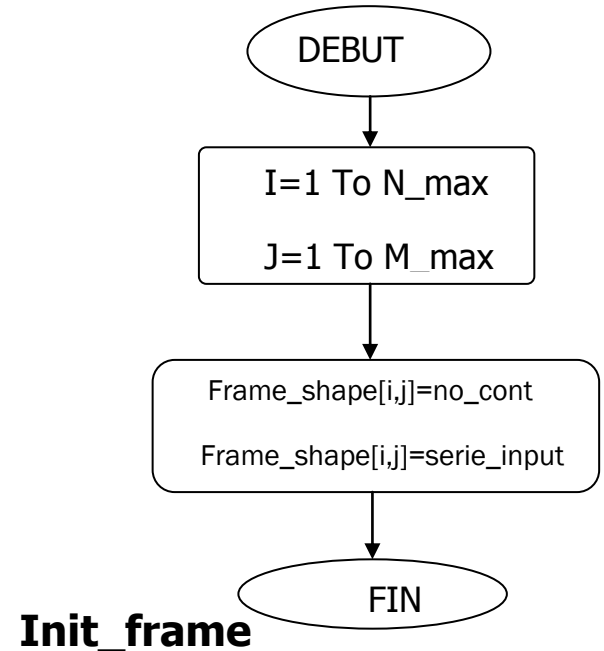
2.Voisinage de OR :



Voisinage_OR

3.Initialisation du Frame :

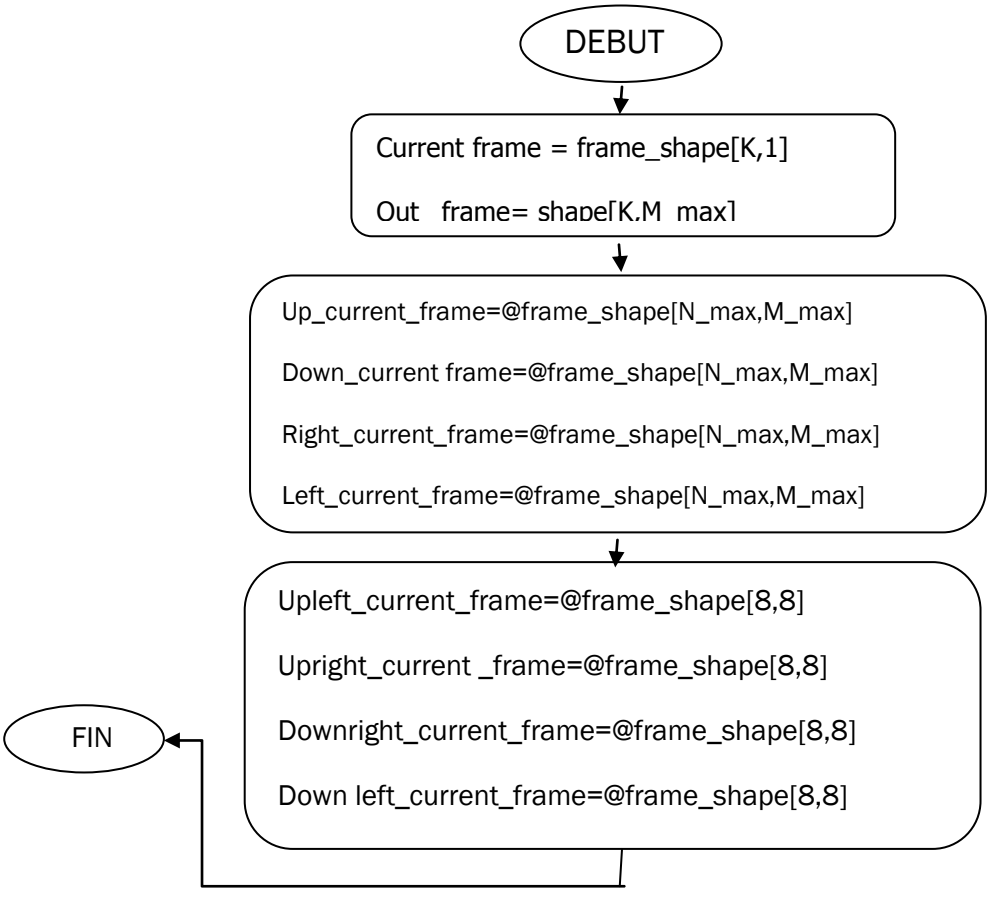
Ce programme initialise le frame courant ou selectionné(current frame)



Init_frame

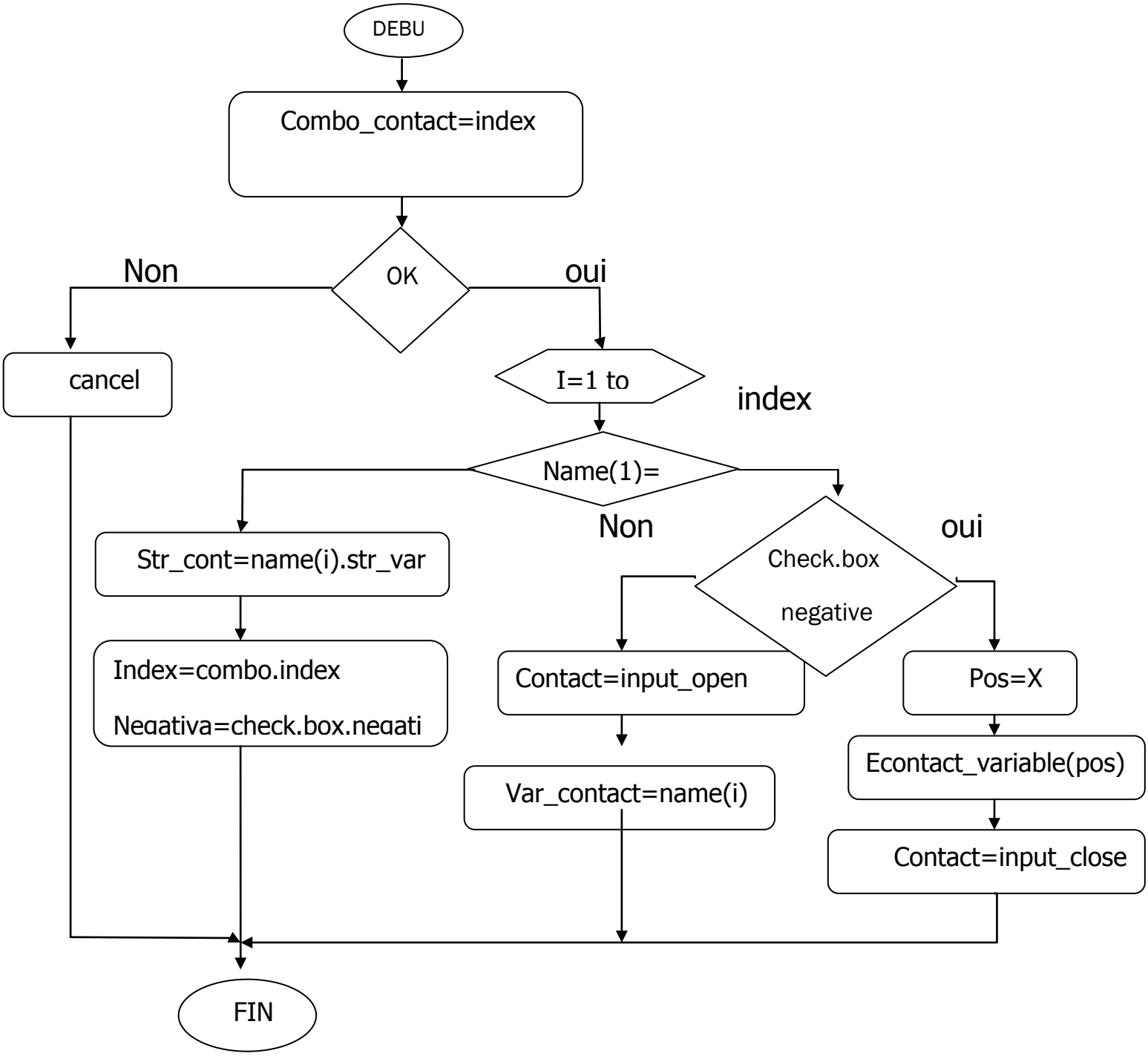
4.Initialisation du voisinage de frame :

Ce programme permet d'initialiser les frames qui entourent le frame sélectionné à l'aide du pointeur qu'on a symbolisé par @



5.Organigramme de << frame double click>> :

Ce programme fait appel à la boite de dialogue là ou on caracterisé le contact en agissant sur le composants places dessus.

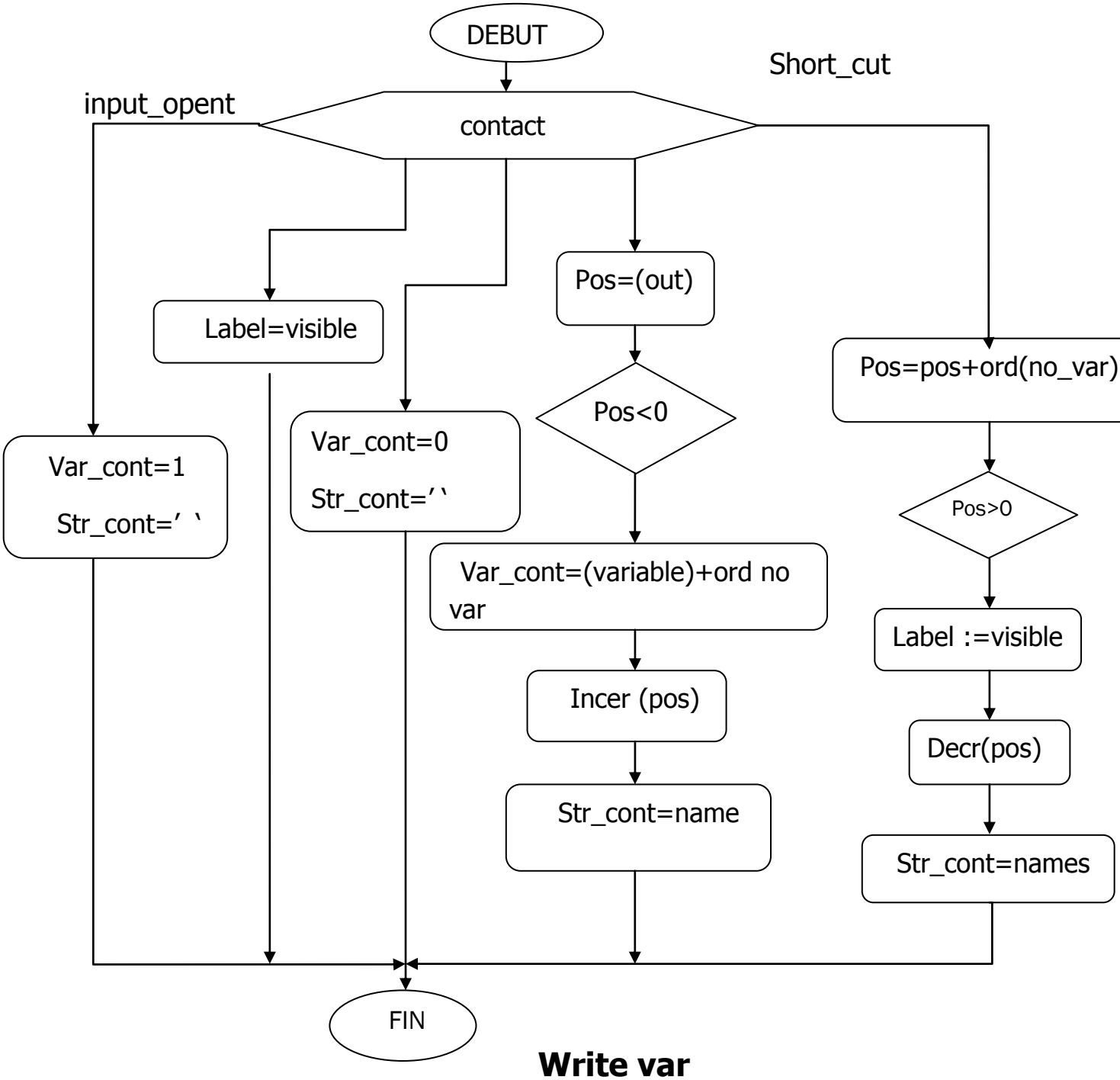


frame double click

Frame double click

6.Organigramme <<Write Var>> :

Le role de cet oranigramme est de permettre a l'utilisateur de donner une variable au contact insérè sur la boite de dialogue



Write var

Les symbolisations appliquées dans le code source de l'application :

1. Econtact: ensemble des contactes (contacte ouvert, contacte fermé....).

2. EIO: ensemble des types des contactes insérés (Noinput, input, output).

3. Enoeud: ensemble des noeuds (noeud série ou parallèle).

4.Eoperateur: ensemble des opérateurs (AND, OR, NOT).

5.Econtact_variable: type énuméré des variables affectées aux contactes insérés. 6. str_var: de type string.

7. Enu_var: de type Econtact_variable

8. Var_contact: de type Econtact_variable.

9. nœud: de type Enoeud

10. names: de type string.

11. TEnsemble Variables = set of EContact Variables;

12. TEnsembleNoeud = set of ENoeud;

13. SContact Variables = record Str_Var: string, Enu_Var: EContact Variables; end;

14. SVar Noeud-record

15. FusionVar_Noeud-record

16. Names_Contacts = table SContact Variables;

17. EMat_var_noeud =table de SVar_Noeud;

18. EMat_Fusion_var_noeud =table de Fusion Var_Noeud;

19. EMat_Fusion_var_noeudX =table de FusionVar_Noeud;

20. EEquivalent VarX-table de Fusion Var_Noeud ;